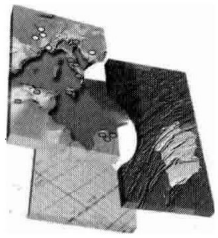
ONWORD PRESS
THOMSON LEARNING

GIS

Developing GIS Solutions with MapObjects™ and Visual Basic®

Bruce A. Ralston

Learn to Program GIS Applications from the Ground Up



Developing GIS Solutions with MapObjects™ and Visual Basic®

Bruce A. Ralston

ONWORD PRESS
THOMSON LEARNING™

Australia Canada Mexico Singapore United Kingdom United States

ONWORD PRESS
THOMSON LEARNING

Developing GIS Solutions with MapObjects™ and Visual Basic®

By Bruce A. Ralston

Publisher:
Alar Elken

Executive Editor:
Sandy Clark

Acquisitions Editor:
James Gish

Managing Editor:
Carol Leyba

Development Editor:
Daril Bentley

Editorial Assistant:
Jaimie Wetzel

Executive Marketing Manager:
Maura Theriault

Executive Production Manager:
Mary Ellen Black

Production Manager:
Larry Main

Manufacturing Coordinator:
Betsy Hough

Technology Project Manager:
David Porush

Cover Design:
Cammi Noah

Printed in Canada
10 9 8 7 6 5 4 3 2 1

For more information, contact:
OnWord Press
An imprint of Thomson Learning
Box 15-015
Albany, New York 12212-15015

Or find us on the World Wide Web at
http://www.onwordpress.com

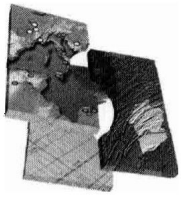
Library of Congress Cataloging-in-Publication Data
is available for this title
ISBN: 0-7668-5438-8

About the Author

Dr. Bruce Ralston is a professor in and head of the Department of Geography at the University of Tennessee, Knoxville. He holds multiple awards, including the Edward L. Ullman Award for Outstanding Contributions to the Field of Transportation Geography, and the Applied Geography Award, both from the Association of American Geographers. He is also a consultant to various public and private sector organizations and President of GIS Tools, Inc.

Acknowledgments

I want to thank the many students who had the desire to learn this material. One of the luckiest things that can happen to a professor is to have students whose enthusiasm for learning pushes him to also learn. I have been extremely fortunate at the University of Tennessee to have a steady stream of such students. They have helped me learn many new areas, including the topics covered in this book. Grateful acknowledgment is also made to the team at OnWord Press.



Introduction

If you have ever tried to learn a foreign language, you know how helpful a dictionary can be. You also know that it is virtually impossible to master a language simply by studying a dictionary or taking classes. Ultimately, you have to immerse yourself in the culture, finding yourself in situations in which you have to use the language to be successful, such as ordering a meal or obtaining directions. A dictionary can help, but it is not enough.

Purpose and Approach

The aforementioned is the problem with help files and, quite frankly, a lot of programming books. They are more like dictionaries. They help you look up specific things once you are conversant in the language, but they do not put you into the culture. Even the example programs in the MapObjects Help file, although useful, provide only "snippets of conversations," not a complete discourse in the original language.

This book attempts to "get you into" the Visual Basic/MapObjects culture. The book begins by examining some simple programs, and by Chapter 4 you will be building a program—a conversation with a user—that becomes more complex in subsequent chapters. In most chapters you will learn a new aspect of Visual Basic programming and then use that knowledge to exploit the potential of MapObjects.

Audience and Prerequisites

This book was developed for users of GIS who want to learn how to develop their own applications using Visual Basic and MapObjects. Although basic aspects of Visual Basic programming are pre-

sented in the text, this is not meant to be a comprehensive introduction to computer programming. The goal of the book is to help readers learn enough about VB and MO so that they can develop their own programs and implement them as either stand-alone applications or as the basis for web-based GIS. Experience with other ESRI products, such as Arc or ArcView, is helpful.

There are some software prerequisites for using this book. Readers should have access to Visual Basic 6, MapObjects 2.0 or later, and (if you wish to implement the web-based application described in Chapter 15) MapObjects IMS. Readers who do not have these ESRI products can download 90-day evaluation copies at *www.esri.com/software/mapobjects/download.html* and *www.esri.com/software/mapobjects/ims/eval.html.* Chapter 14 requires the user to have a web server application that supports PERL scripts.

How To Use This Book

The book contains a series of example programs that build from a very simple map display application in Chapter 4 to advanced applications in later chapters. Starting with Chapter 4, each succeeding chapter will add new functionality to the program developed in the previous chapters. Each program developed in the text can be found on the companion CD-ROM, so that readers can either enter the code that is described in the text or load the corresponding sample programs from the companion CD-ROM.

Content and Structure

This book can be broken into four parts. Chapters 1 through 3 provide a brief introduction to programming with Active X components and Visual Basic (VB). Readers familiar with VB may wish to skim these chapters. The remaining chapters deal with developing GIS programs. In each chapter, a new aspect of VB is discussed, along with new GIS properties that can be found in MapObjects (MO). The VB discussed in the chapter is then combined with MO objects to add more GIS functionality to the program developed in the chapters. Chapters 4 through 7 deal with managing map layers.

Chapter 7 deals with identifying elements of map layers and working with selected sets. Chapters 8 through 11 cover thematic mapping. Chapter 12 deals with collections, classes, and theme-on-theme selections. Chapters 13 through 15 cover web-based GIS. Chapter 13 provides a brief introduction to HTML, whereas Chapters 14 and 15 present two approaches to serving maps on the Web. The final three chapters (16 through 18) deal with map projections, coordinate systems, and buffering and overlay.

Book Features and Conventions

This edition includes a companion CD-ROM at the back of the book (see "About the Companion CD-ROM" at the end of this introduction). Throughout the book you will see references to the companion CD-ROM. It is there that you will find VB projects, data sets, DLLs, and other utilities discussed in the text.

Italic font in regular text is used to distinguish certain command names, code elements, file names, directory and path names, user input, and similar items. Italic is also used to highlight terms and for emphasis.

The following is an example of the monospaced font used for code examples (i.e., command statements) and computer/operating system responses, as well as passages of programming script.

```
var myimage = InternetExplorer ? parent.
cell : parent.document.embeds[0];
```

The following are the design conventions used for various "working parts" of the text. In addition to these, you will find that the text incorporates many exercises and examples.

NOTE: *Information on features and tasks that requires emphasis or that is not immediately obvious appears in notes.*

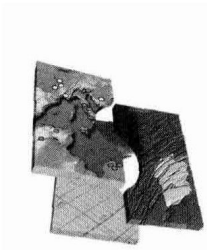
TIP: *Tips on command usage, shortcuts, and other information aimed at saving you time and work appear like this.*

CD-ROM NOTE: *These notes point to files and directories on the companion CD-ROM that supplement the text via visual examples and further information on a particular topic.*

About the Companion CD-ROM

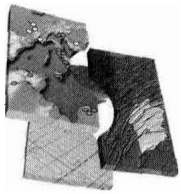
The companion CD-ROM consists of four directories, each of which contains material used in the text. Each directory and its content are described in the following.

- *MO2*: This directory contains all VB projects referenced in the text. Whenever you see a CD-ROM Note that references a particular directory, it will be found under this directory. These projects can be opened and executed in Visual Basic.
- *Utility*: This directory contains a DLL for converting bitmap files to *jpg* files. It is used in Chapter 14 to develop web-based GIS without a commercial IMS product.
- *Web*: This directory contains materials used for web-based GIS. These include HTML pages, two PERL scripts, Active Server Pages (ASP) scripts, and a *jpg* file of Tennessee. The use of these files is discussed in Chapters 13 through 15.
- *Shapes*: The shapes used in the examples are stored here. There are two directories under the *Shapes* directory. *USA* contains shapes for the United States, and *Knox* contains shapes for Knox County, Tennessee. The shapes in the *USA* subdirectory are used in Chapters 4 through 15, and the shapes in the *Knox* subdirectory are used in chapters 16 through 18.



Contents

Chapter 1



Entering the VB/MO Culture

Introduction

Developing GIS solutions requires writing code that can manage and manipulate geographic data. Advances in software development tools have accompanied the rapid acceptance of GIS. Two advances in software development have been the development of object-oriented programming (OOP) languages and the use of components for the distribution of objects that can be used with OOP languages. Visual Basic (VB) is a language that allows users to develop and manipulate objects. MapObjects contains a set of objects of interest to GIS application developers. Together, these tools make it possible to develop OOP-based GIS implementations.

An Introduction to OOP Implementations

To understand OOP you must understand objects. Objects have *properties* and *methods*. It is important that you understand the difference between these. First, however, you need to understand the concepts "class" and "object." A *class* is an abstract (meaning non-discrete) term, such as *human being*. An *object* is a discrete realization or instance of a class (i.e., by analogy, a particular human being). It is in the class that *properties* and *methods* are defined. Every object (and therefore its properties and methods) is the realization of a class, just as every person is the realization of the concept "human being."

Toward understanding the difference between properties and methods, let's use another real-world analogy: a bicycle. A bicycle (an "object") has many properties, among them color, size, weight, number of tires, brand, and type of spokes. Applying this analogy to scripting, if you wanted to determine a property (such as color or brand) of the object (bicycle), you would typically use a *Get* command, as in

```
color = bicycle.GetColor()
```

or

```
brand = bicycle.GetBrand()
```

The first instance returns a color, which might be represented by a string (such as BLUE) or a number (such as 0xFF000, which is the hex number—a number in base 16—for orange). The second instance is more likely to return information represented as a string, such as TREK (a brand of bicycle). The point is: *You must know what type of variable your* Get *request is returning.* At this point you might want to try adjunct exercise 1-1, which follows.

Adjunct Exercise 1-1: Determining Variables Returned by the Get Request

***NOTE**: This exercise assumes you have ArcView 3.*

To determine variables returned by the *Get* request, perform the following steps.

1. Start ArcView.
2. Go to the help file, click on the INDEX tab, and type *GET.*
3. Scroll down and note that there are a lot of entries. Click on the FIND tab, and type *get* (lowercase, as this is case sensitive).

How many "get" topics have been found? Not all of these will be for *Get* requests, but, as you saw from the INDEX tab, a lot of them will be.

You now know that you can find an object's properties (like a map's extent or projection, or a layer's default symbol) with a *Get* request. Curiously, however, not all *Get* requests use the word *Get*!

Returning to our bicycle example, suppose you were ordering a new bike. In this case, you might want to order a specific color. The sister command of *Get* is *Set*, as in

```
bicycle.SetColor(BLUE)
```

Here, the color of the bicycle has been set to *BLUE*. Note that there is no equals sign (or more accurately "assignment operator") in this statement. (At this point you might want to try adjunct exercise 1-2, which follows.)

Adjunct Exercise 1-2: Determining Variables Returned by the Set Request

***NOTE**: This exercise assumes you have ArcView 3.*

To determine variables returned by the *Set* request, perform the following steps.

1. Start ArcView.
2. Go to the help file, click on the INDEX tab, and type *SET*.
3. Scroll down and note that there are a lot of entries. Click on the FIND tab, and type *set* (lowercase, as this is case sensitive).

How many "set" topics have been found? Not all of these will be for *Set* requests, but, as you saw from the INDEX tab, a lot of them will be.

One last thing about *Set*: so far you have been looking at OOP languages in the abstract. Later in the book, you will see in regard to VB that there are two related ways of setting values: *Set* and *Let*. You do not need to worry about the difference between these just yet.

Some properties cannot be changed. For example, in regard to the bicycle example, you might not allow the following:

```
bicycle.SetNumTires(3)
```

Bicycles, by definition, must have two (and not more or less than two) tires. If there are three tires, it is a tricycle, not a bicycle! (One tire, and it is a unicycle.) This points out a very important rule, expressed in the following Note.

***NOTE**: With some properties you have read (*Get*) and write (*Set*) access. With others, you only have read (*Get*) access. Still others only have write (*Set*) access.*

Finally, consider the tires themselves. They, too, are objects. They, too, have properties and methods. When you want to inflate the tires, the object of interest is the tire, not the bicycle. That is,

```
bicyle.SetPressure(90)
```

may be illegal, but

```
tires.SetPressure(90)
```

may be fine.

This points out yet another important lesson, expressed in the following Note.

***NOTE**: You must know which properties and methods go with which object.*

This is often confusing and seemingly counterintuitive, but it is extremely important. Consider another example. Suppose you had an object named *HOUSE*. The house object contains several other objects, one of which might be *BATHTUB*. A command such as

```
bathtub.SetWaterLevel(top)
```

might be fine, but

```
house.SetWaterLevel(top)
```

might be disastrous! However,

```
bathtub.SetCleanToday(TRUE)
```

and

```
house.SetCleanToday(TRUE)
```

might both be legal and desirable.

In summary, you know that there are things called objects, which have properties and that can consist of other objects. You can determine the value of those properties with *Get* commands, and may be able to change those properties with *Set* commands.

Let's turn to methods. Methods are the things objects do. For example, a "person" object might have "eat," "sleep," "move," and "rest" methods. Our "bicycle" object might have "move" and

"stop" methods. A map object can be drawn or scaled. A layer object can be turned off or on. In fact, you have already encountered methods. For example, to redraw a map in VB/MO, you would issue the command *themap.Refresh.*

Working in Visual Basic

Let's take a look at VB. As stated, VB uses an integrated development environment (IDE) that facilitates a "trial-and-error" approach to program development. That is, VB programs are developed in the IDE and can be tested without compiling the program or leaving the IDE. To start the IDE, you need only start VB (see figure 1-1).

CD-ROM NOTE: *The VB project in the* Chapter 1_1 *directory on the companion CD-ROM starts here.*

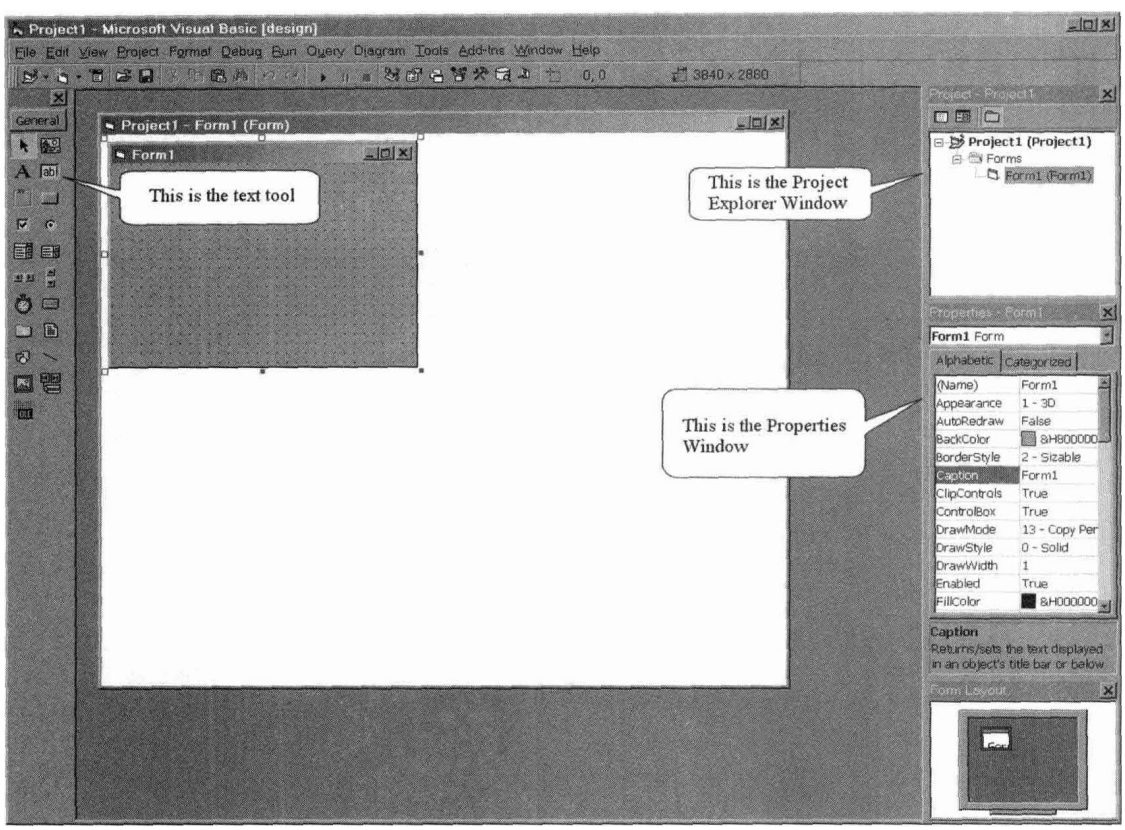


Fig. 1-1. The Integrated Development Environment (IDE).