Wrox Programmer to Programmer™

Professional

Microsoft® Robotics Developer Studio

Kyle Johns, Trevor Taylor

WROX™

Updates, source code, and Wrox technical support at **www.wrox.com**

Professional Microsoft® Robotics Developer Studio

Kyle Johns
Trevor Taylor

WILEY
Wiley Publishing, Inc.

Professional Microsoft® Robotics Developer Studio

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published simultaneously in Canada

ISBN: 978-0-470-14107-6

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data:

Johns, Kyle, 1965-
Professional Microsoft robotics developer studio / Kyle Johns, Trevor Taylor.
p. cm.
Includes index.
ISBN 978-0-470-14107-6 (paper/website)
1. Robotics. I. Taylor, Trevor, 1955- II. Title.
TJ211.J55 2008
629.8'9—dc22

2008014648

To Ryan, Tanner, Kaitlyn, Kelsey, Colin, and little Abby; but most of all to Marie.
—Kyle

To Denise, whose support enabled me to undertake writing this book.
—Trevor

About the Authors

Kyle Johns is a principal software developer at Microsoft, where he is currently a member of the Microsoft Robotics Developer Studio Team. After receiving a master's degree in computer science from the University of Utah, he designed 3D graphics hardware for flight simulators at Evans and Sutherland. He joined Microsoft as one of the original members of the DirectX Team and then went on to help develop the graphics system software in the early days of the Xbox project. Recently he has been enjoying the opportunity to apply his 20 years of 3D graphics experience to the field of robotics by developing the Robotics Developer Studio Simulation Environment.

Trevor Taylor is a consultant in the field of robotics education. After 20 years in the IT industry, including co-founding a consulting company that became a Microsoft Solution Provider Partner, he moved to the Queensland University of Technology in 2002. For six years he taught a variety of subjects, including Visual Basic and Web development using ASP.NET. During this period he also worked part-time on a doctorate in computer vision and robotics. In early 2008, Trevor left QUT to concentrate on developing course materials for teaching robotics and to finish writing his thesis. Trevor has worked with MRDS since the very first Community Technology Preview in June 2006 and is an active and well-known contributor to the community.

Credits

Executive Editor
Chris Webb

Development Editor
Maureen Spears

Technical Editors
David E. Buckley
Steve Tudor

Production Editor
William A. Barton

Copy Editor
Luann Rouff

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Joseph B. Wikert

Project Coordinator, Cover
Lynsey Stanford

Proofreaders
Jeremy Bagai,
David Fine,
Paul Sagan,
Word One

Indexer
Robert Swanson

Acknowledgments

Various organizations have assisted with the production of this book by making software services for their robots available for inclusion in the book examples and/or providing images for use in the figures. The authors wish to acknowledge their support. These organizations include (in alphabetical order) the following:

CoroWare, Inc.: `www.coroware.com`

Institute for Personal Robots in Education (IPRE): `www.ipre.org`

LEGO: `www.lego.com`

Lynxmotion, Inc.: `www.lynxmotion.com`

Parallax, Inc.: `www.parallax.com`

Picblok Corporation Pty Ltd: `www.picblokcorporation.com`

RoboticsConnection (Summerour Robotics Corporation): `www.roboticsconnection.com`

Surveyor Corporation: `www.surveyor.com`

Throughout the preparation of the book, the Microsoft Robotics Developer Studio Team have been most helpful in resolving problems and answering questions. To them we extend a big "thank-you."

Foreword

I am pleased to have the opportunity to introduce this book and Microsoft Robotics Developer Studio to its audience. I hope that the combination helps foster the creation of interesting applications for robots because it is clear that the key to the success of this emerging new industry is software that delivers value.

Anyone who has been around for a while recognizes that the current robotics industry has many similarities to the early PC industry. Both are characterized by both the tremendous passion and anticipation of the early pioneers and questions about the value of the technology. With the advent of word processing, spreadsheets, and thousands of other applications, no one asks me any longer why I own a PC. In fact, most of them also own PCs. Similarly, the personal robotics market that is just emerging has the same or even greater potential if the creativity of its community can be unlocked.

It is the vision and energy of the existing robotics community that directly contributed to Microsoft investing in the creation of Microsoft Robotics Developer Studio. While many employees at Microsoft have had a personal interest in robotics, it was not until the leadership from diverse parts of the robotics community invited Microsoft to participate that the company decided to get started. Until that time, robotics developers were considered part of an audience consisting of casual, academic, and professional developers who might use Visual Studio and our embedded operating systems.

In 2004, things began to change. I happened to be in the right place at the right time to help make that happen. At the time, I was working in the offices of Microsoft's chief software architect, aka Bill Gates. Part of my job was to promote synergies across the company, as well as to be an extra set of eyes and ears for Bill. That enabled me to attend a number of meetings with notable members of the robotics community whose projects ranged from educational robotics to academic research and from consumer robotics to industrial automation. Despite the diversity of the audience, these people all reflected a common message — something significant was happening; and they encouraged Microsoft to participate and apply its software expertise just as it had in the past to PCs and the Web.

I began a dialogue with Bill about how Microsoft might respond to this invitation. The discussion was very timely. Bill, having recently visited a number of universities to speak on the importance of computer science to future technology, was treated to several campus tours that included some innovative work in robotics. This enabled him to see firsthand the tremendous progress and investments being made in the robotics community. As a result, he asked me to spend the next few months gathering more data and formulating a specific proposal about what we might do.

I then talked with even more people in the robotics community, including Red Whittaker from CMU, Sebastian Thrun from Stanford, and Rod Brooks from MIT, to name just a few. All of them confirmed my earlier impressions that the market for robotics was primed to move forward in a new, more personal dimension. However, they also indicated that significant challenges were holding back its progress. Specifically, every robot was an island unto itself, resulting in limited sharing of technology. Developing software for robots has historically required a great deal of technical expertise and resources because the tools and technologies needed have been typically tied to specific hardware platforms. Therefore, despite a growing interest, it was hard and often expensive for anyone to participate.

I compiled my research into a 60-page proposal outlining how Microsoft could help address these challenges, which I sent to Bill, who requested that I review it with Craig Mundie, who serves as chief technology officer focused on forward-looking strategies for the company, and Rick Rashid, Microsoft's senior vice president of research, to get their insights and feedback from the perspectives of the academic and research communities. Both executives were very supportive. Craig offered some advanced technologies he had been incubating and Rick offered to host the new incubation.

From there it took about nine months to prototype what would become Microsoft Robotics Developer Studio. After another executive review, the prototype was approved, leading to a new product and business for Microsoft. In June 2006, the product was formally previewed; it was released in December that same year.

What was delivered is an impressive collection of software. It starts with the runtime, which provides a very simple, but flexible model for development. The Concurrency and Coordination Runtime (CCR) and Decentralized Software Services (DSS) libraries were originally intended as a future programming model to enable developers to write applications for the increasingly prevalent multi-processor, distributed scenarios into which computing technology is evolving. CCR enables developers to move beyond the limitations of single-threaded applications without the conventional complexities of programming locks and semaphores. DSS takes that simple programming model and applies it across the network, providing a simple yet elegant, flexible model that can run on a single PC or across networked servers. Modeled on conventional Web-style interfaces such as HTTP and SOAP, it extends the so-called Representational State Transfer (REST) model by providing a powerful notification (publish-subscribe) framework that makes the system very efficient, yet applicable to both Windows and web applications.

Furthermore, the architectural framework provides a better model for resiliency and software maintenance. Software modules, which we refer to as services, operate in isolation from each other, both from a data sharing model as well as execution standpoint. This means that failures (bugs) in one service are unlikely to corrupt the data or execution of other services. In addition, individual services can be shut down, restarted, or replaced dynamically, meaning software maintenance can be performed while an application is running. Because data sharing is done through message-passing and is represented as documents, it is an easy model for maintaining the integrity of the data shared between models. This code-data separation also makes it possible to easily create different views or user interfaces of the data without affecting the code creating the data. Therefore, it is possible to have many different ways to visualize the data. In fact, one of the easiest ways to view the state of a service is to simply use a web browser because each service state has a URL.

Finally, the CCR/DSS programming model provides for rich *composability* — that is, applications can be composed of functions provided by other modules. This is reflected in the number of generic services that are included with Microsoft Robotics Developer Studio. It enables specialization of services at one level, such as a specific motor, while enabling more advanced services, such as a drive service, to define partner contracts that enable it to operate with a variety of different motor hardware. This capability to apply series across different robots, results in not only greater opportunity for accessing enabling technologies, but also a larger market for those creating those technologies.

In addition to the programming model afforded by the CCR and DSS runtimes, we added tools to make it easier to develop applications. While supporting a wide variety of programming languages, including C#, C++, Visual Basic, and Python, we added the Visual Programming Language (VPL), which enables an easy drag-and-drop approach to creating applications. This tool not only provides a somewhat easier way to create applications, but also facilitates rapid prototyping because it offers the optional generation

of human-readable C# code. Moreover, because of the common services-oriented programming model, the language is infinitely extensible.

The Visual Simulation Environment provided also makes development easier. Its 3D graphical presentation is augmented with a software physics engine for a realistic emulation of robots and their environment. Because it uses the same services framework, it is possible to develop applications in simulation and easily move them to the real hardware, which often saves time and resources. Services can be created that run without regard to whether they apply to the simulation or the real world. The software physics engine means that interaction between simulated entities is freely available. For example, when two entities collide, the simulation automatically applies the result both operationally and visually.

Finally, we added a great deal of sample code — in terms of robot models, enabling technologies, and tutorials — to the toolkit. The objective is to help people get started. Most of these samples are provided in both compiled and source form, so every sample can serve as a learning experience. Of course, with so many samples and tutorials, getting started may sometimes be overwhelming.

That's where this book comes in. It provides a number of additional ways to explore beyond the samples in the toolkit and therefore is a good companion for developers just beginning. It also illustrates creative ways to use the tools and libraries in Microsoft Robotics Developer Studio to build innovative new robotics applications.

I believe that by enabling a wider audience of contributors personal robotics has the potential to become as much a reality as personal computing has become today. I hope that Microsoft Robotics Developer Studio and this book provide important catalysts toward that end.

Tandy Trower
General Manager
Microsoft Robotics Group

Introduction

Microsoft Robotics Developer Studio (MRDS) introduces a new way to program robots in the Windows environment. It attempts to bring some order to the chaos that has marred the field of robotics, at least as far as Windows-based applications are concerned. If it meets its objectives, you will see the emergence of a common standard for robotics software in the next few years. This is potentially very significant, and it is a great time to get in on the ground floor.

MRDS was developed over a relatively short period of time by a very small team. It is not your average Microsoft product. In an unusual move for Microsoft, key portions of the code for MRDS are available in source form. This makes it readily extensible and offers many opportunities for programmers, whether you are a hobbyist, a research student, or an employee of a large corporation, to write new services that integrate directly into the system and to easily share these services with others.

The speed of development and the limited resources of the MRDS team has meant that documentation has tended to lag behind. This book provides a comprehensive introduction to MRDS for experienced C# programmers. It contains a wealth of examples. We did not set out to duplicate the existing MRDS documentation, but instead to complement it.

Because programming with MRDS is so fundamentally different from what you might have been used to in the past, getting started can be difficult. This is where a good textbook with well-constructed examples can be useful. When this book was first proposed, no such textbook was available.

In the spirit of sharing, all of the source code for this book is available online. The examples address many of the key issues involved in building robotics applications, and we hope that they will act as launching pads for many more new and exciting services. In the years to come, we hope to look back at this book and laugh at how primitive the examples were.

Kyle has extensive experience in simulation and games development, and he is the MRDS team member responsible for the MRDS simulator. Trevor has worked with robots and computer vision for many years and has been active in the MRDS community since the very first Community Technology Preview in 2006. We have a passion for robotics and we have endeavored to pass on our knowledge through this book.

Many readers may be wondering why the examples are only in C#. The answer is simple: Weigh this book. Adding VB would double that, adding C++ would triple it, and so on. It was simply not feasible to offer a multi-language version of the book in its first edition.

We welcome feedback on the book and suggestions for further examples. The book's website (`www.wrox.com`) provides a feedback mechanism. You can also get updates at `www.proMRDS.com`. VS 2008 versions of the code will be posted to this site, as well as updates when MRDS V2.0 is officially released. In addition, the MRDS Discussion Forum hosts an active community that is constantly exchanging ideas and code, so you are encouraged to participate in the Forum.

If we compare the current state of development in robotics to the release of the IBM PC in the 1980s, then we have an exciting and challenging time ahead. Welcome to the field of robotics!

Who Is This Book For?

This book is for programmers who want to learn about the rapidly growing field of robotics. The primary focus of the book is on developing software to control robots, and many of the examples are designed to work with the MRDS simulator. However, you can't call yourself a roboticist unless you have worked with some real robots, so also included are examples using a wide variety of different robot platforms.

We assume that you have prior programming experience with C#, the preferred language for MRDS. Although you can use other .NET languages with MRDS, all of the examples in this book are in C#. If you do not know C#, don't let this deter you, especially if you have experience in Java or C++.

Although the book is intended to be used by both professional programmers and undergraduate students, it is not aimed at beginning programmers.

What Is This Book's Focus?

MRDS is a framework for developing software to control robots. Therefore, the book emphasizes the programming techniques and patterns that you use to write MRDS services.

A major advantage of the approach taken in the book is that you can get started without buying any hardware because MRDS includes an excellent simulator. The simulation examples, and the examples for VPL (Visual Programming Language), can be run without real robots.

The book is not intended to replace the large amount of documentation that is already available for MRDS. Instead, it attempts to lead you through various aspects of MRDS by way of examples.

Both of the authors have extensive experience in using MRDS. All of the sample services are complete and based on the authors' experience.

What Does This Book Cover?

The book is divided into four parts. You should work through Part I first because it contains all the background information and concepts that are crucial to understand MRDS. After that, you can jump to any of the remaining parts because they are designed to be relatively independent of each other.

Part I: Robotics Developer Studio Fundamentals

The book begins with a brief overview of MRDS in Chapter 1. It includes instructions for installing MRDS and setting up the sample code from the book that is available for download from `www.wrox.com`. Some quick examples enable you to verify that the code is working, and you get a first taste of MRDS. You will find references to a variety of resources at the end of the chapter that will be helpful as you learn MRDS.

Chapter 2 launches into MRDS concepts with a range of examples demonstrating how to program in the multi-threaded environment provided by the Concurrency and Coordination Runtime (CCR). The CCR presents a new programming paradigm, but it is key to the power of MRDS. Concurrency is essential to robotics, where often many tasks must be performed simultaneously. Consider a robot driving around on its own — sensors must be read to detect obstacles; steering decisions have to be made in real time; and motors need to be controlled. Chapter 2 sticks to basic CCR constructs and you do not need a robot to follow along. The chapter concludes with a list of fundamental code patterns that can be used in MRDS programming.

MRDS programs are written as services, which run under the control of Decentralized Software Services (DSS), so Chapter 3 explains the important features of services and how to write them. This chapter may present a lot of information that is new to you. Several topics are addressed, such as starting and stopping services; configuring services; and how to package your services for deployment. The examples demonstrate how two services can communicate with each other, and again no robots are required.

In Chapter 4, you work through examples of common tasks, such as subscribing to other services to receive notifications and adding a user interface using either Windows Forms or Web Forms. This chapter also discusses abstract services, known in MRDS as *generic contracts*. The sample services, Dashboard and TeleOperation, enable you to control a variety of robots, either real or simulated. These examples include a wide range of useful code patterns, including how to use a web camera as an input device. In this chapter, you finally use a real robot!

Part II: Simulations

In Chapter 5, the MRDS simulator is introduced with an example that shows off the physics features of the simulator. The various features of the simulator are demonstrated, including the Simulation Editor. The robot models and other environmental entities provided with the simulator are also presented. A good simulator is invaluable for prototyping services, and it is well worth spending time becoming familiar with the simulator. The built-in robot simulations enable you to test ideas quickly.

Chapter 6 moves on to developing new simulation entities. A four-wheeled robot, based on the Corobot from CoroWare, is created, along with a simulated infrared range sensor. In addition to the simulation, you learn how to build a new service that complies with an existing generic contract — the Quad Differential Drive is compatible with the generic (two-wheeled) Differential Drive. Even if you do not intend to build your own simulated robot, the new simulation entities in this chapter might be useful in your own projects.

Based on the Corobot in Chapter 6, the code in Chapter 7 implements a simulated Robo-Magellan competition similar to the one run by the Seattle Robotics Society. The robot must navigate around a course using a simulated GPS to locate a set of markers. It coordinates with a referee service that times how long it takes the robot to locate the targets. The SimMagellan service combines computer vision with a wandering behavior and obstacle avoidance to successfully navigate the course and identify the markers. This is a great example to use as a jumping-off point for more complex and, it is hoped, smarter behaviors.

Up to this point, the book concentrates on wheeled robots. In Chapter 8, a simulated robotic arm is built that is based on the Lynxmotion L6 arm, which has six degrees of freedom. Later, in Chapter 15, a real robotic arm is used and the two match closely enough that the same services can be used with either. In building the simulated arm, basic concepts are covered, including joints, coordinate frames, and

inverse kinematics. Some pre-defined procedures, such as stacking dominos, put the arm through its paces and are fascinating to watch.

Chapter 9 brings together a variety of different simulations, including robot soccer; a hexapod (six-legged walking robot); a Maze Simulator; and ExplorerSim. The ExplorerSim service uses a simulated Pioneer 3DX to explore a maze using a laser range finder and to build a map as it goes. The services in this chapter are not only good examples of how to write simulations, they are also useful in designing and testing your own applications. For example, the Maze Simulator can be used with your own services so that you can test algorithms for exploration, maze solving, and other applications of artificial intelligence.

Part III: Visual Programming Language (VPL)

The basics of the Visual Programming Language (VPL) are covered in Chapter 10. It begins by outlining the concepts behind a data flow language and builds the obligatory "Hello World" example, but with a twist — this one talks to you. Then it moves on to discuss variables, loops, custom activities (the rough equivalent of subroutines), and some simple debugging. The chapter is rounded out with information on using lists and the `Switch` activity.

In Chapter 11, more advanced topics are discussed such as processing sensor data and controlling robots. If you have a real robot, such as the LEGO NXT, you can use it for some of the examples. However, the examples were written using the MRDS simulator so that you do not need a robot. Instructions are also provided for creating C# code from a VPL diagram and compiling it into a service. This facility is handy if you want to prototype in VPL and then hand-optimize the code later or add features that are easier to implement directly in C#. The last part of the chapter touches on controlling multiple robots with a couple of different examples.

Chapter 12 provides several VPL examples. The VPLExplorer service performs a similar function to the ExplorerSim service introduced in Chapter 9. The robotic arm makes a reappearance with a service that can be used to control the arm using an Xbox controller. With only minor changes, the VPL diagram can be made to work with either a simulated arm or a real robotic arm. Line-following sensors are constructed in the simulator to enable an iRobot Create robot to follow a line on the ground. Lastly, simple computer vision is used to enable a simulated Corobot to follow a ball.

Part IV: Robotics Hardware

The variety of different robots that are supported under MRDS is outlined in Chapter 13. Some advice is given on selecting a robot if you plan to buy one. This type of information is a moving target — it changes on an almost daily basis. You should check the Microsoft MRDS website and the book's website (`www.proMRDS.com`) for updates. The chapter then explains some basic terminology and provides background information on robotics. This coverage is intended to ensure that everyone is up to speed with all the buzzwords before continuing with the rest of the chapters. No new services are introduced.

Chapter 14 builds services for the LEGO NXT and the Parallax Boe-Bot. Updated services for the Boe-Bot are included in this chapter that provide additional functionality to the services supplied by Parallax. The discussion in this chapter focuses on remotely controlled robots, so Bluetooth figures prominently. A simple "dance" program exercises the robots, and just by changing the manifest, the same program is run using simulated robots. An extended online version of this service is available that handles multiple

robots simultaneously. Wandering behavior using sensors is developed in the last part of the chapter and implemented on both of the real robots using their specific capabilities, rather than generic contracts.

The Lynx 6 arm makes a comeback in Chapter 15, and forward and inverse kinematics, which control the arm motions, are discussed. Revised versions of the Lynxmotion services are provided that incorporate new kinematics algorithms. A service to record arm motions is developed as an example of the two fundamentally different approaches to controlling an arm — moving individual joints or setting the position of the end effector (the tip of the arm) using 3D coordinates. In this case, the reverse process now applies and the simulated arm can be used with the service developed for the real arm.

In Chapter 16, the wonderful world of autonomous robots is introduced. Up to this point in the book, the robots are all controlled remotely. Now the "brains" are moved onto the robots. Two different approaches are explored: an embedded PC and a PDA (personal digital assistant). These devices run a slimmed-down version of Windows known as Windows CE or Windows Mobile. In this environment you have to use the .NET Compact Framework (CF), which necessitates some changes to how you write services. A Stinger robot is set up to wander autonomously using first a PDA mounted on top of the robot and then an eBox 2300 embedded PC.

Finally, in Chapter 17, entirely new services are built from scratch. A new generic contract is defined for a generic brick (robot brain) that encapsulates the common elements of many different low-end robots as far as sensors and actuators are concerned. A test service is built to enable easy testing of services based on the generic brick. Using this new contract, services are implemented for the Picblok Integrator and the K-Team Hemisson robots. The onboard monitor program (firmware) for the Integrator's PICAXE chip is developed in this chapter as well as the MRDS services. Calibration of sensors and drive motors is covered using the Hemisson robot as an example.

What You Need to Run the Examples

As pointed out earlier, you do not need a real robot to use much of the code in this book because you can use the simulator in MRDS. The book has been deliberately designed so that people on a tight budget, such as students, can learn the fundamentals of MRDS without real robot hardware.

If you do want to work with a real robot, you should read Chapter 13, which discusses the available robots. The robots selected for use in this book cost only a couple of hundred dollars and are well within the reach of the average professional programmer. They are small enough to use in an office environment and are readily available — most can be ordered online. If you have a bit more money to spend, the principles in the book apply equally to robots costing many thousands of dollars.

To run the examples, you should remember the following:

- The Microsoft Robotics Developer Studio development environment runs on Windows XP (Service Pack 2) or Vista. It requires the .NET Framework V2.0 Service Pack 1. The simulator requires a machine with a graphics card that supports DirectX 9 with support for 2.0 Pixel and Vertex Shader programs.
- All the code in this book was written using the Version 1.5 Refresh of MRDS, released in late December 2007. The MRDS software can be downloaded from the Microsoft website and is free of charge for noncommercial use. The MRDS kit automatically installs .NET 3.0, DirectX 9.0, XNA 1.0, and the AGEIA PhysX engine 2.7, which the MRDS simulator uses.

- You need Visual Studio 2005 to compile code for this release of MRDS — either VS2005 Professional or the Express Edition of C#, available as a free download from the Microsoft website. It is advisable to apply Visual Studio Service Pack 1. Visual Studio 2005 automatically installs .NET V2.0, but you must apply the .NET Service Pack 1.

The code has not been tested using Visual Studio 2008, which was released while the book was being written. Visit the book's website (`www.proMRDS.com`) for the latest information on using Visual Studio 2008 with MRDS and the book examples.

- Up to four gigabytes of disk space is required to install the full Visual Studio 2005 Professional with the MSDN Library. The Express Edition of C# does not require as much space, but it cannot be used to develop mobile (CF) applications such as those in Chapter 16.
- It is recommended that you have at least 512MB of memory on your PC. If you intend to use the Visual Programming Language or the simulator, you should have a minimum of 1GB of memory or your system will page heavily and run very slowly.
- Services can be built that target Windows Mobile or Windows CE devices such as personal digital assistants or embedded PCs. For the examples in Chapter 16 you need a mobile device.
- Rather than provide a CD with the book that would quickly become outdated, the source code for all of the examples in the book is available for download from the Wrox website (`www.wrox.com`), as described in the Source Code section that follows. As new versions of MRDS are released by Microsoft, you will also be able to download updated versions of the sample code from `www.proMRDS.com`.

After the 1.5 Refresh version of Robotics Developer Studio was released, Microsoft changed the name of the development kit to Microsoft Robotics Developer Studio. We have used this new name and the abbreviation MRDS in most places in the book.

Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

Code has several styles. When we're talking about a word in the text — for example, when discussing the `helloForm` — it appears in `this font`.

```
This text is used to show regular code blocks.
In code examples we highlight changes to previous examples or important code with a
gray background.
```

Sometimes you will see a mixture of styles. For example, where a section of code shows commands that you should type, the code you type is presented in **bold**:

```
C:\Microsoft Robotics Studio (1.5)>dssnewservice
```

> **Boxes like this one hold important, not-to-be forgotten information that is directly relevant to the surrounding text.**

Tips, hints, tricks, and asides to the current discussion are offset and placed in italics like this.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- Keyboard strokes are shown like this: Ctrl+A.
- We show filenames, URLs, and code within the text like so: `index.html`
- We indicate a break in a line of code with the ➥ symbol.

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for download at `www.wrox.com`. Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-14107-6.

The book code is supplied in DssDeploy format, which is a self-expanding executable. Alternately, you can go to the main Wrox code download page at `www.wrox.com/dynamic/books/download.aspx` to see the code available for this book and all other Wrox books.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or a faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration and at the same time you will be helping us provide even higher quality information.

To find the errata page for this book, go to `www.wrox.com` and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page you can view all errata that have been submitted for this book and posted by Wrox editors. A complete book list, including links to each book's errata, is also available at `www.wrox.com/misc-pages/booklist.shtml`.

If you don't spot "your" error on the Book Errata page, go to `www.wrox.com/contact/techsupport.shtml` and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.