

A-PDF Text Replace DEMO: Purchase from www.A-PDF.com



Professional
**Crystal Reports® for
Visual Studio® .NET
2nd Edition**

David McAmis



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

Professional Crystal Reports®

for Visual Studio® .NET

Second Edition

David McAmis



Wiley Publishing, Inc.

Professional Crystal Reports®

for Visual Studio® .NET

Second Edition

David McAmis



Wiley Publishing, Inc.

Vice President and Executive Group Publisher: Richard Swadley

Vice President and Executive Publisher: Bob Ipsen

Vice President and Publisher: Joseph B. Wikert

Executive Editorial Director: Mary Bednarek

Senior Acquisitions Editor: Jim Minatel

Editorial Manager: Kathryn A. Malm

Senior Production Editor: Fred Bernardi

Development Editor: Adaobi Obi Tulton

Production Editor: Felicia Robinson

Media Development Specialist: Kit Malone

Text Design & Composition: Wiley Composition Services

Copyright © 2004 by Wiley Publishing, Inc., Indianapolis, Indiana. All rights reserved.

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4447, E-mail: permcoordinator@wiley.com.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEB SITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEB SITE MAY PROVIDE OR THE RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEB SITES LISTED IN THIS WORK MIGHT HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993, or fax (317) 572-4002.

Trademarks: Wiley, the Wiley Publishing logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates. Crystal Reports is a registered trademark of Seagate Software, Inc. Visual Studio is a registered trademark of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Control Number: 2004003621

ISBN: 0-7645-5730-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

*To Tony Breese, who is still keeping the faith,
waiting for that sci-fi book.*

Acknowledgments

As with any book, there was incredible team of people who brought this project to life. First and foremost, thanks to Sharon Cox for guiding this book through the transition to Wiley. And special thanks to Adaobi Obi Tulton, who was the glue that held this project together and who had to put up with me for so long! To everyone else from the Wiley/Wrox team who was involved in the book, a heart-felt thank you.

Behind the scenes, there are many people I work with, especially Colin, Alice, and Lisa, who have been a sounding board and have had to hear about this project for going on two years now. As always, a big thank you needs to go out to the Crystal Decisions Asia Pacific team for their support and assistance, and to Craig in particular for his faith and confidence in me and for continually keeping me challenged.

And on a personal note, there is a worldwide network of friends and family who offer their support and encouragement on a daily basis, which I couldn't do without. *Nunc scio quia sit amor.*

About the Author

David McAmis is a Crystal Certified Consultant and trainer, living and working in Sydney, Australia as a partner in Avantis Information Systems. As a consultant for Avantis and on behalf of Crystal Decisions, David creates Windows, Web, and mobile applications incorporating Crystal technology across a wide number of platforms, databases, and ERP systems to deliver innovative solutions for common business problems.

In his varied career, he has held the roles of consultant, technical trainer, university lecturer, and consulting services manager and has served as vice-president of a software and services company in the United States. David holds a B.S. degree in Management Information Systems and is a Microsoft Certified Professional, as well as a certified trainer and consultant for numerous software products.

David has been working with Crystal Reports since version 4.5 and is an active member of the beta and user group community and a self-confessed “raving fan.” In his career as a Crystal developer and trainer, he has traveled the world and taught over 800 students. You can reach him at dmcamis@hotmail.com.

Contents

Acknowledgments	v
Introduction	xv
<hr/>	
Chapter 1: Crystal Reports .NET Overview	1
 What Is Crystal Reports?	2
A Brief History	3
 What Can You Do with Crystal Reports .NET?	3
 How Is Crystal Reports .NET Different from Previous Versions of Crystal Reports?	7
Integrated Design Environment	7
Any Language, Any Time	7
Integration Methods	8
Ease of Use	8
Building Enterprise Applications	8
Report Architecture	9
Report Designer	12
Incompatibilities	14
 Crystal Reports .NET Benefits	14
Leverage Existing Development and Skills	14
Tight Visual Studio .NET Integration	15
Windows and Web Report Viewers	15
Easy Deployment	15
ADO .NET	16
XML Report Web Services	16
 Installing Crystal Reports .NET	16
 Learning from Sample Applications	18
Installing Sample Applications	18
Sample Reports	20
Sample Data	21
Tutorials	21
 Summary	23

Contents

Chapter 2: Getting Started with Crystal Reports .NET	25
The Sample Files	26
Planning Your Report Design	27
Creating a Crystal Report	29
Adding a Report to Your Application	30
Using a Report Expert	31
Working with the Report Design Environment	46
Menus and Toolbars	47
Setting Default Properties	48
Report Design Basics	50
Report Sections	50
Report Formatting	52
Field Objects	52
Summary	59
Chapter 3: Designing Reports	61
The Sample Files	61
Creating a New Report	62
Working with Databases and Tables	62
Adding a Data Source to Your Report	64
Setting the Data Source Location	68
Verifying Database Structures	69
Working with Groups	69
Inserting a New Group	70
Changing Groups	72
Deleting Groups	73
Formatting Groups	73
Record Sorting	75
Working with Summaries	76
Inserting a Summary Field	76
Changing a Summary Field	78
TopN/Group Sorting	78
Using Running Totals	80
Using Cross-Tabs	83
Formatting Cross-Tabs	86
Working with Charts	89
Working with Subreports	93
Inserting Subreports	94
Changing Subreports	96
Creating On-Demand Subreports	97
Saving and Reimporting Subreports	100

Working with Parameter Fields	101
Creating a Parameter Field	101
Optimizing Report Performance	104
Summary	105
Chapter 4: Report Integration for Windows-Based Applications	107
Obtaining the Sample Files	107
Planning Your Application	108
Exploring the Development Environment	109
Starting a New Windows Application with VB .NET	110
Determining the Correct Object Model	112
Understanding the CrystalDecisions.Windows.Forms Namespace	113
Using the Crystal Report Viewer for Windows Forms	114
Adding a Report to Your Application	116
Adding the Report Viewer to a Windows Form	117
Binding a Report to the Report Viewer	118
Passing Database Logon Info	121
Setting Report Record Selection	124
Working with Parameter Fields	125
Customizing the Appearance and Behavior of the Report Viewer	130
Viewer Methods	132
Using Viewer Events	140
Drilling into Report Details	143
Drilling Down on Subreports	145
Dealing with Report Exceptions	146
Summary	147
Chapter 5: Report Integration for Web-Based Applications	149
Obtaining the Sample Files	150
Planning Your Application	150
A Brief History of Crystal Web Development	151
Exploring the Development Environment	152
Before You Get Started	152
Starting a New Web Application with VB .NET	153
Determining the Correct Object Model	154
Understanding the CrystalDecisions.Web Namespace	155
Using the Crystal Report Viewer for Web Forms	156
Adding a Report to Your Application	159
Adding the Report Viewer to a Web Form	162
Binding a Report to the Report Viewer	162
Setting Report Record Selection	168
Working with Parameter Fields	169

Contents

Customizing the Appearance and Layout of the Report Viewer	173
Viewer Methods	177
Printing Your Report	182
Using Viewer Events	188
Summary	191
Chapter 6: Creating XML Report Web Services	193
 Obtaining the Sample Files	193
 XML Report Web Services Overview	194
What Are XML Report Web Services?	194
How Would I Use an XML Report Web Service?	196
 Creating XML Report Web Services	196
Creating Basic Report Web Services	197
Creating Report Web Services with Multiple Reports	201
Utilizing the Generic Report Web Service	202
 Consuming XML Report Web Services	204
External Report Web Service	205
Internal Report Web Service	206
Generic Report Web Service	207
 Deployment Considerations	208
 Summary	210
Chapter 7: Working with .NET Data	211
 The Sample Files	211
 Data Access with Crystal Reports .NET	212
Database Files	213
Relational Databases	213
OLAP Data	214
Crystal Dictionaries, Queries, and Info Views	214
Other Datasources	214
 Working with Datasources	215
Setting Database Options	216
Adding a Database or Table to a Report	217
Using the Visual Linking Expert	221
Verifying Database Structures Used in Your Report	224
Changing a Database Location	225
Setting a Database Alias	226
 Working with SQL Commands and Expressions	228
Defining Virtual Tables	228
Creating SQL Expressions	232

Working with ADO .NET	234
An ADO .NET Dataset	234
Viewing the Contents of a Dataset	236
Creating a Report from an ADO .NET Dataset	239
Viewing Reports Containing an ADO .NET Dataset	240
Summary	243
Chapter 8: Formulas and Logic	245
Integrating Formulas and Logic into Your Reports	246
Database Structures	246
Application Data	247
Crystal SQL Commands	247
Crystal SQL Expressions	248
Formulas	249
Working with the Formula Editor	251
Controlling the Editor's Appearance	251
Controlling the Syntax Type	252
Checking for Syntax Errors	253
Creating Formulas with Basic Syntax	255
What Is Basic Syntax?	255
Basic Syntax Coding Conventions	256
Simple Operators	257
Control Structures	266
Creating Formulas with Crystal Syntax	268
Differences from Basic Syntax	268
Creating Record Selection Formulas	269
Working with Conditional Formatting	270
Understanding Conditional Formatting	270
Conditional Formatting for Boolean Properties	270
Conditional Formatting for Multiple-Outcome Properties	271
Summary	273
Chapter 9: Working with the Crystal Reports Engine	275
Obtaining the Sample Files	276
Understanding the CrystalDecisions.CrystalReports.Engine Namespace	276
Customizing Reports Using the Report Engine	276
Getting Started	277
Printing and Exporting	279
Working with Databases	285

Contents

Working with Areas and Sections	294
Working with Report Objects	297
Customizing Report Fields at Run Time	306
Summary	309
Chapter 10: Distributing Your Application	311
Distribution Overview	311
Getting Started	312
Setup Projects	312
Web Setup Projects	312
Merge Module Project	313
Setup Wizard	313
Basic Deployment Requirements	313
Operating System	314
Hardware	314
Deploying Windows Applications	315
Creating a New Setup Project	315
Selecting Project Outputs	317
Determining Run-Time File Requirements	318
Adding Merge Modules	319
Working with Licensing	321
Building Your Setup Project	322
Testing and Deploying Your Setup Project	322
Deploying Web Applications	322
Preparing Your Web Server	323
Creating the Setup Project	323
Building Your Setup Project	324
Testing and Deploying Your Setup	324
Summary	325
Appendix A: Troubleshooting	327
Appendix B: Migrating Applications to Crystal Reports .NET 2003	335
Appendix C: Crystal Syntax versus Basic Syntax	339
Index	345

Introduction

Welcome to the second edition of *Professional Crystal Reports for Visual Studio .NET*. Crystal Reports is one of the world's leading software packages for creating interactive reports that can be integrated into a wide range of Windows and Web applications. With more than four million licenses shipped, Crystal Reports is the leader among Windows report writers. Crystal Reports has been in the Visual Studio box since 1993, but with the introduction of Visual Studio .NET 2002, a new version, Crystal Reports .NET, was integrated more closely than ever before. And with the release of Visual Studio .NET 2003, the bar has been raised even higher, with tighter integration and more reporting options. This book will detail the functionality provided with Crystal Reports for Visual Studio .NET 2003 and how, when, and where you should integrate reports into your .NET applications.

Why incorporate reports into applications? Virtually all applications need to present data to users, but any work beyond basic formatting—charts or conditional formatting, for example—can be very complex to program manually. In this book, we will provide you with the practical, high value, real-world information that you need to understand the array of tools that Crystal Reports for Visual Studio .NET provides for developers, so that you can immediately begin creating rich reports that can be integrated into your Windows- and Web-based applications.

This book does not attempt to be all-inclusive, and it will not teach basic .NET techniques. To be able to deliver a functional guide to Crystal Reports for Visual Studio .NET, we assume that you have a grasp of essential programming techniques—in this case, a knowledge of programming in Visual Basic .NET and experience using Visual Studio .NET—and that you can apply these skills to a new technology.

Examples are carefully chosen to demonstrate the capabilities of Crystal Reports for Visual Studio .NET and aid you in understanding the techniques that you can apply when you begin to use this technology in your .NET applications.

Who This Book Is For

This book is for programmers who want a comprehensive guide to the functionality included with Crystal Reports for Visual Studio .NET. It's assumed that you have some knowledge of .NET and experience with Visual Studio .NET.

This book is mainly aimed at readers who have some experience with Crystal Reports. However, this edition has been specifically updated to include a large section on report design, which can assist new users in learning how to create their own reports. This title should also prove valuable for readers who are new to reporting in general and who want a guide to this free reporting tool they've discovered within Visual Studio .NET.

What This Book Covers

This book covers the features of Crystal Reports for Visual Studio .NET that you'll find yourself using time and again to build complex reports and integrate them into different .NET applications. We start by explaining how Crystal Reports fits into the .NET platform and how it differs from previous versions of Crystal Reports. Then, we discuss the key techniques we can use:

- ❑ Creating reports using the Expert
- ❑ Employing advanced report design topics
- ❑ Integrating reports into Windows- and Web-based applications
- ❑ Creating XML Report Web Services
- ❑ Working with ADO .NET
- ❑ Using formulas and logic in our reports
- ❑ Developing distributed reporting applications
- ❑ Deploying our applications

How This Book Is Structured

We begin with an overview of Crystal Reports for Visual Studio .NET, introducing the technology, what we can use it for, and the benefits of integrating Crystal Reports with the .NET Framework. We then go on to create some reports and to learn how to integrate them into both Windows- and Web-based applications. We examine XML Web Services and how to work with ADO .NET and formulas and logic in our reports, and then finish by developing distributed reporting applications and looking at how to deploy the applications we have created throughout the book.

The text is organized into 10 chapters:

Chapter 1: Crystal Reports .NET Overview—In this chapter, we take our first look at Crystal Reports for Visual Studio .NET (Crystal Reports .NET), including how the product is different from other versions of Crystal Reports, how to find and run the sample applications that are included, and where to find the tutorials that will get you up to speed with the product. We will also take a look at the new Crystal Reports .NET architecture and learn how it fits in to the .NET Framework. Whether you are an experienced application developer looking to move to Visual Studio .NET or you are developing your first application and have never heard of Crystal Reports, it all starts here.

Chapter 2: Getting Started with Crystal Reports .NET—In this chapter, we will be looking at the Crystal Reports Designer within Visual Studio .NET and learning how to create and import reports for use in Windows or Web applications. By the end of the chapter, we will have the skills to develop our own basic reports and will be able to move on to the actual integration of these reports into our application. If you have used Crystal Reports before, some of the material in this chapter will be familiar.

Chapter 3: Designing Reports—In this chapter, we will be designing reports using features found in the Crystal Reports Designer within Visual Studio .NET. By the end of the chapter, we will have the skills to create complex reports that incorporate the most popular Crystal Reports features.

Chapter 4: Report Integration for Windows-Based Applications—In this chapter, we are going to look at how to integrate and view the reports that we created in the last chapter from Windows applications, and we will see how to customize our reports at run time using the rich object models provided. Throughout the chapter we will be looking at code examples to illustrate the use of various features. By the end of the chapter, we should be familiar with the majority of report integration concepts and be ready to apply them to our own application development.

Chapter 5: Report Integration for Web-Based Applications—In this chapter, we are going to look at how to integrate and view reports from within Web-based applications created with Visual Studio .NET. In addition, we will look at some of the run-time customizations that can be made to our reports, as well as some issues around Web-application deployment. As we go through this chapter, we will be building forms for use in Web-based reporting applications, which demonstrate many of the features of Crystal Reports .NET that can be used in our own Web applications.

Chapter 6: Creating XML Report Web Services—We have now seen how to integrate reports into Windows- and Web-based applications, but now we need to learn how to leverage those skills and work with XML Report Web Services. This chapter will teach us to identify what an XML Report Web Service is and to understand how it can be used in our application. We will also create a Report Service from an existing Crystal Report and utilize the service with the Crystal Windows or Web Viewer.

Chapter 7: Working with .NET Data—In this chapter we take a step back to look at what lies underneath the reports we have created—the data our report is based on and how Crystal Reports .NET uses this data. We will look at the way Crystal Reports works with different data sources and how it interacts with ADO .NET. At the end of this chapter, we will have an understanding of how Crystal Reports .NET interacts with different datasources, the options for working with these datasources, and how to use ADO .NET as a datasource for our report development.

Chapter 8: Formulas and Logic—This chapter will narrow our focus to look at where the majority of Crystal Reports development time is spent: writing formulas and logic. We will discover the best way to add calculations and logic to our reports and learn enough syntax and code to handle most situations. We will also see how to differentiate between the two different flavors of the Crystal Formula Language and how to write our own record selection and conditional formatting formulas.

Chapter 9: Working with the Crystal Reports Engine—In this chapter, we will be looking at the Crystal Reports Engine, the functionality it provides, and some of the advanced integration techniques that we can use in our own application. We learn to identify when to use the Crystal Reports Engine namespace, how to integrate it into our application, and how to use the features contained within the properties, methods, and events associated with the engine.

Chapter 10: Distributing Your Application—Finally, with our development and testing finished, we will, in this chapter, look at one of the last steps in the software development life cycle—the actual deployment of our application to the end users. We will examine the tools Visual Studio .NET provides to help distribute applications and how these tools can be used to distribute applications that integrate Crystal Reports. This chapter has been designed so that if you are interested only in deploying Windows applications, you can turn immediately to that section and get started. Likewise, if you are developing Web applications, there is a separate section for Web deployment. By the end of this chapter, we will be able to identify the set-up and distribution tools within Visual Studio .NET and to understand how they can be used to package and distribute our application. We will also be able to create a set-up package from an application that integrates Crystal Reports and to successfully install it on a target machine.

What You Need to Use This Book

There are software and knowledge requirements for successful progress through this book:

Software

- Microsoft Windows 2000 or XP Professional
- Visual Studio .NET 2003 Professional or higher
- SQL Server 2000 or MSDE

Knowledge

- Some knowledge of the Visual Studio .NET 2003 development environment is assumed
- Some very basic knowledge of SQL is assumed

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.

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* important words when we introduce them
- We show keyboard strokes like this: Ctrl-A
- We show file names, URLs, and code within the text like so: `persistence.properties`
- We present code in two different ways:

In code examples we highlight new and important code with a gray background.

The gray highlighting is not used for code that's less important in the present context or that has been shown before.

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; for this book the ISBN is 0-764-55370-0.

Once you download the code, just decompress it with your favorite compression tool. 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. This book assumes all of the code is located on your C:\ drive, so the path of the examples described in the book will begin C:\Crystal.NET2003\ChapterXX\, where XX is the chapter number. If you extract the code to somewhere other than the C:\ drive, please keep this in mind when you are working with the examples.

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, like a spelling mistake or 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 has 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.

p2p.wrox.com

For author and peer discussion, join the P2P forums at <http://p2p.wrox.com>. The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and to interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At <http://p2p.wrox.com> you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

- 1.** Go to <http://p2p.wrox.com> and click the Register link.
- 2.** Read the terms of use and click Agree.

Introduction

- 3.** Complete the required information to join, as well as any optional information you wish to provide, and click Submit.
- 4.** You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1

Crystal Reports .NET Overview

Crystal Reports has enjoyed a long association with Microsoft and has shipped with Visual Basic (and subsequently Visual Studio) as the default report writer since 1993. Developers have traditionally had a love-hate relationship with Crystal Reports; they loved the functionality it provided and the free run-time license, but they hated having to upgrade to the latest version to get the features they required. Another complaint was that reports could not be created or modified programmatically; they could be created only through the user interface (UI) with either the developer UI with Visual Studio or the consumer UI with the Crystal Reports retail package.

Just as the release of Visual Studio .NET 2002 represented a significant leap for the Microsoft development platform, the release of Crystal Reports for Visual Studio .NET was also a milestone for the Crystal Decisions development team. Following the Microsoft .NET strategy, they redeveloped the product to take advantage of the .NET Framework and made it a fully featured product in its own right; developers no longer have to wait to upgrade to the latest release to get the features they need.

When Visual Studio was upgraded recently to Visual Studio .NET 2003, the version of Crystal Reports that ships inside the box was also updated. In this chapter, we are going to take a first look at Crystal Reports for Visual Studio .NET 2003 (Crystal Reports .NET), examining how the product is different from other versions of Crystal Reports, how to find and run the sample applications that are included, and where to find the tutorials that will get you up to speed with the product. We will also take a look at the Crystal Reports .NET architecture and learn how it fits into the .NET Framework.

Whether you are an experienced application developer looking to move to Visual Studio .NET or you are developing your first application and have never heard of Crystal Reports, it all starts here.

What Is Crystal Reports?

In simplest terms, Crystal Reports is a report design tool that allows you to create reports capable of retrieving and formatting a result set from a database or other data source. In addition to simply reading data from a data source, Crystal Reports has its own formula language for creating calculations and includes a number of features that can be used to turn raw data into presentation-quality reports, with graphs, charts, running totals, and so on.

If you look at all of the different types of reports that can be created using Crystal Reports, shown in Figure 1-1, you will find that they are as varied as the developer or end user who created them.

You can create reports that range from a simple list with only a few columns to a complex management report that shows multiple graphs, tables, and Key Performance Indicators (KPIs). The flexibility of the report designer means that it can be used for many different types of output, depending on your needs.

In addition to a powerful toolset for creating reports, Crystal Reports also features a number of Application Programming Interfaces (APIs) and tools specifically created for developers to allow them to integrate these reports into their own applications. To help understand these features and how they are used, we are going to have a brief look at the history of the products leading up to this release of Crystal Reports .NET.



Figure 1-1

A Brief History

In the beginning, a small company in Vancouver called Crystal Services developed a DOS-based reporting add-on for ACCPAC accounting in 1988. A few years later, in 1992, the company released Crystal Reports, touting it as the “world’s first Windows report writer,” and it wasn’t too long after that Microsoft standardized Crystal Reports as the reporting engine for Visual Basic. The rest is history.

Within a year of that historic partnership between Crystal Services and Microsoft, over a million licenses of Crystal Reports were shipped, giving it a foothold within the developer community and ensuring its long-term success. Since that time, Crystal Reports has evolved alongside the available platforms and development tools, moving from floppy distribution to CDs, from 16- to 32-bit, and from a .dll print engine to ActiveX control to embedded designer to Automation Engine to .NET Classes.

Over the years, through the transition of the company from Crystal Services to Seagate Software to Business Objects, the user interface for creating reports hasn’t changed much; the basic features are still the same, even though the look and feel of the icons and menu bars may change depending on the UI design standards of the day. What have really changed over the years and releases of Crystal Reports are the functions and features that have been developed, culminating in a product that can easily hold its own with just about every other report writer on the market. To have a look at some of those features, we are going to delve into exactly what you can do with Crystal Reports .NET.

What Can You Do with Crystal Reports .NET?

To start with, Crystal Reports .NET includes an integrated Report Designer available within the Visual Studio IDE (shown in Figure 1-2) that you can use to create report files (.rpt) to integrate with your application.

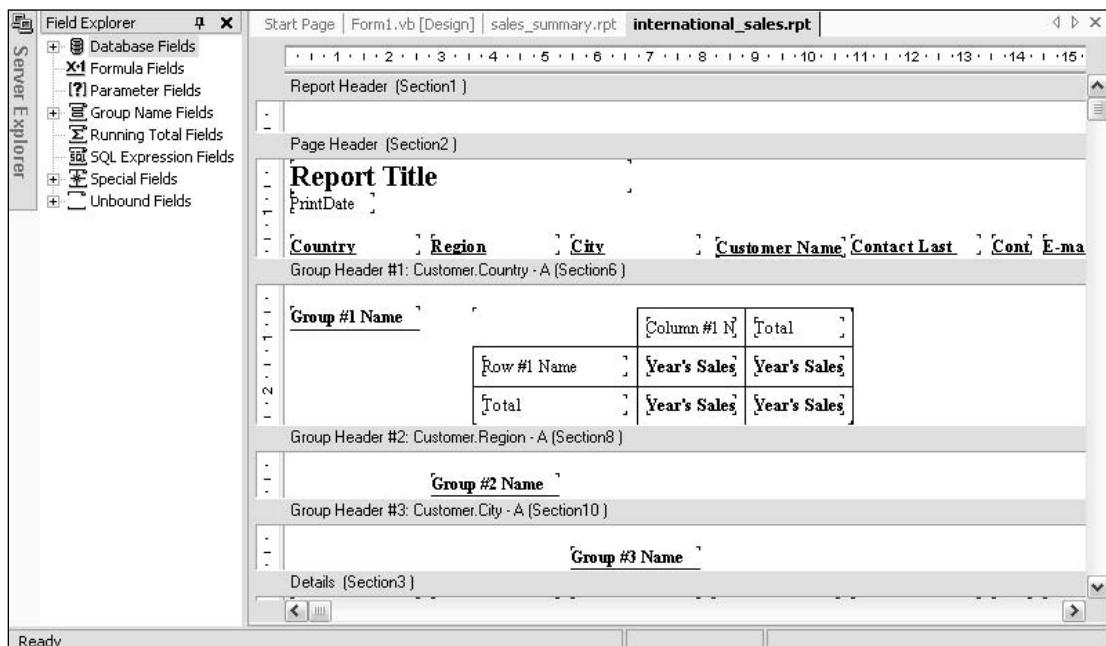


Figure 1-2

Chapter 1

This Report Designer (covered in Chapter 3, “Designing Reports”) features a number of experts (or wizards) to help you get started creating a report. It will guide you through the report development process, from selecting a data source and the field that will appear on your report, to determining what records should appear.

Once you have a basic report designed, you can then add features like formula fields, running totals, graphs, and so on to make your report design as complex as required. Reports come in all shapes, sizes, and forms. You may want to create a report that can be used to print an invoice from your application, to compile statistics for a management report, or to produce an inventory count sheet.

You don’t even have to constrict yourself to a particular size or shape; reports can be created that print shipping labels or address labels and can include bar codes, pictures, graphics, and so on.

To get an idea of the types of reports that can be created using Crystal Reports, check out the sample reports available from the Crystal Decisions Web site at <http://community.crystaldecisions.com/fix/samplescr.asp>.

After you have created a report, you need some way to display it from your application. Crystal Reports .NET has two different viewers to make this happen. The Windows Forms Viewer (which we look at in Chapter 4, “Report Integration for Windows-Based Applications”) can be used with Windows applications to preview any reports you have integrated into your application. It features a rich object model that allows you to control the appearance of the viewer and some aspects of the report at run time.

You can add this viewer to any form in your application, either as the sole content of the form or as one of several form components. You can control the viewer’s appearance, changing toolbars, and other visual aspects, even creating your own icons and buttons to control the viewer and its actions, like the viewer shown in Figure 1-3.

For Web-based applications, there is also a Web Forms Viewer (Chapter 5, “Report Integration for Web-Based Applications”) that has similar functionality and allows you to view reports you have integrated into your Web applications. You can add this viewer to Web pages within your application and show a report either on its own page, in a frameset, or like the report in Figure 1-4, side by side with other application content; it is up to you.

For complete control over your report, regardless of whether you are viewing it through the Windows or Web Forms Viewers, you also have access to the Report Engine (see Chapter 9, “Working with the Crystal Reports Engine”). This will allow you to control even the most minute aspect of your report before you view it using one of the aforementioned viewers. Using the Report Engine, you can control the report’s formatting and features, set database credentials, and call direct methods to print, export, and so on.

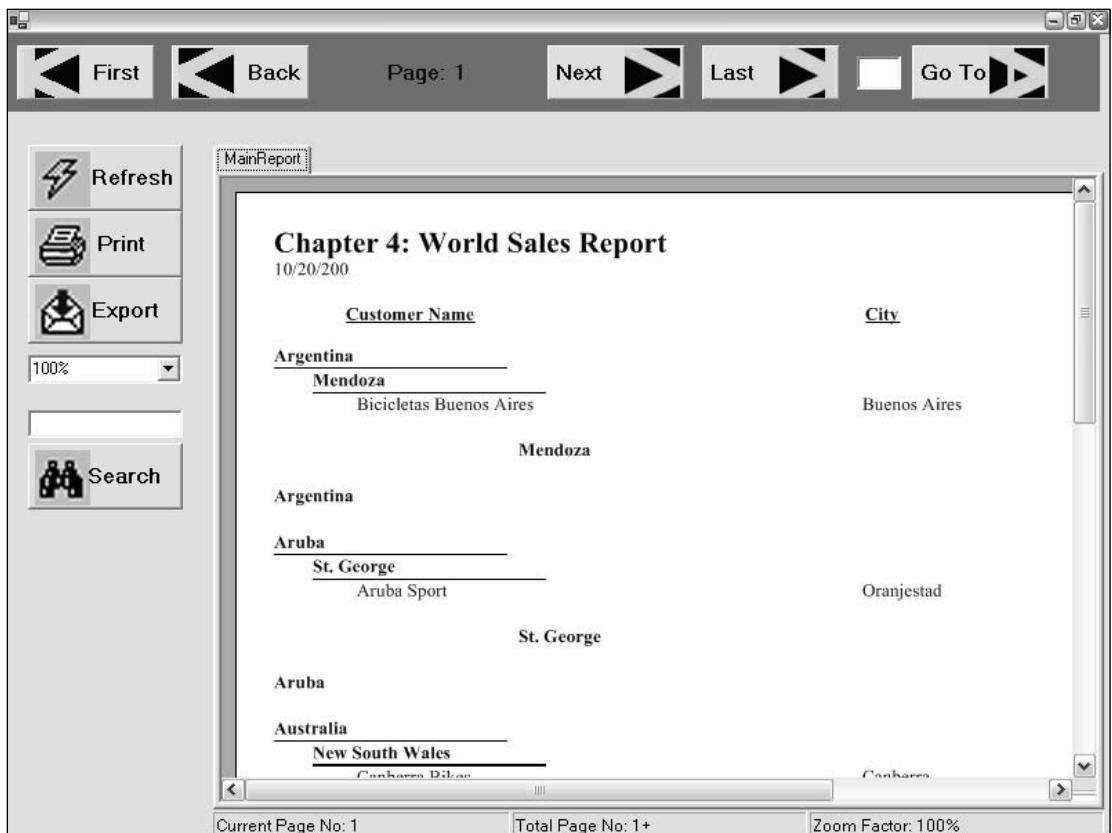


Figure 1-3

For creating distributed applications, Crystal Reports .NET has a number of features specifically designed for creating and consuming XML Report Web Services, either through the generic Web Service that ships with Crystal Reports .NET (which allows you to utilize a report without having to publish it as a Web Service) or by creating your own Web Services from report files, like the one shown in Figure 1-5. In any case, Chapter 6, “Creating XML Report Web Services” will guide you through the process of both creating and consuming XML Report Web Services.

Chapter 1

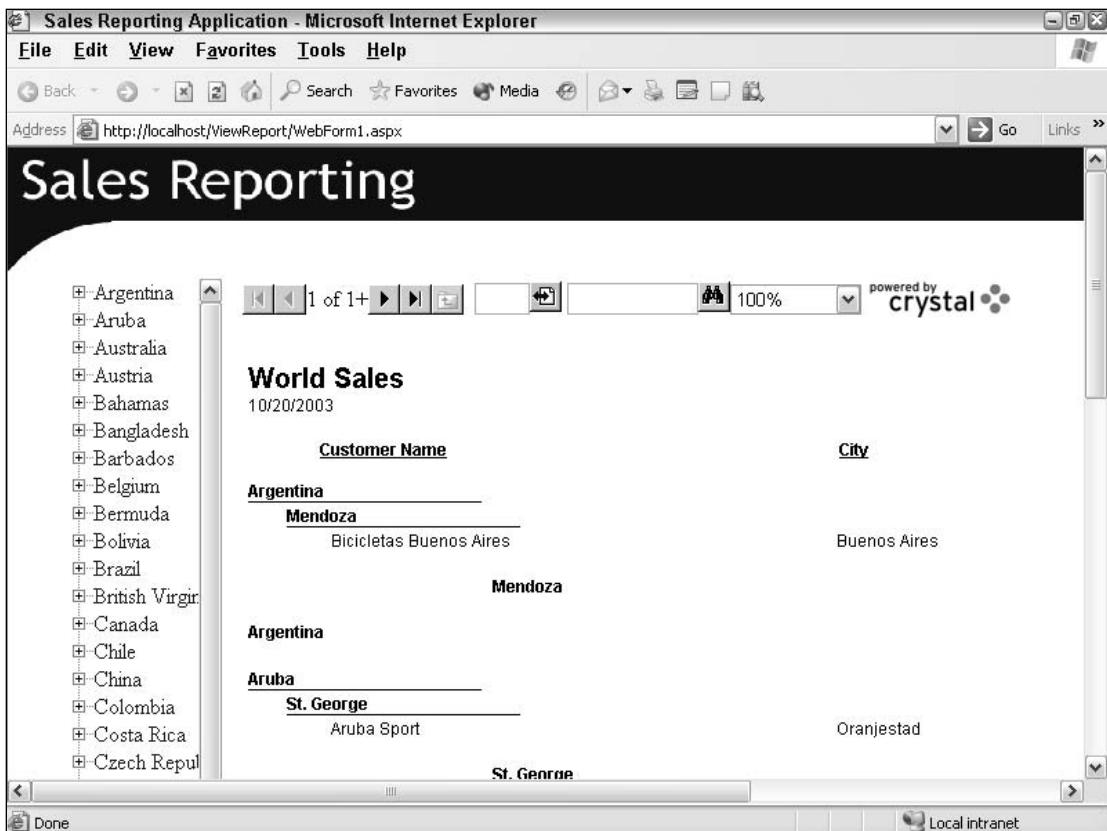


Figure 1-4



Figure 1-5

Crystal Reports .NET is also tightly integrated with Crystal Enterprise, a report scheduling and distribution system that provides a true multi-tier back-end processing platform for reports and allows you to use a scheduling engine and distribution framework to distribute reports to thousands of users.

And finally, there are a number of tools that have been included for distributing reports with your application, including updated merge files for this version of Crystal Reports. In Chapter 10, “Distributing Your Application” we will look at how to use these tools to successfully deploy your own applications and how to trouble-shoot installation and setup problems.

How Is Crystal Reports .NET Different from Previous Versions of Crystal Reports?

Crystal Reports .NET 2003 is an updated version of the report writer integrated with Visual Studio .NET 2003 and has been updated from the original version that first shipped with Visual Studio .NET 2002. This version is a special OEM version of Crystal Reports that is available with the Visual Studio .NET suite. It shares some common features with the retail version of Crystal Reports and was built on the Crystal Reports 8.x technology, but components of Crystal Reports .NET have been rewritten using C# and are designed to take full advantage of the .NET Framework.

Integrated Design Environment

Unlike the standalone versions of Crystal Reports, Crystal Reports .NET is part of the Visual Studio .NET Integrated Development Environment (IDE). Using the integrated Report Designer, you can create or modify reports from within the Visual Studio .NET IDE. If you have used the Report Design component from previous versions of Crystal Reports, the concept will be familiar.

Any Language, Any Time

Crystal Reports .NET follows the Visual Studio .NET mantra of “any language, any time” and is not too picky about the language you use to write reporting applications. You can use any of the .NET languages (VB, C#, J#, C++, and so on) to develop reporting applications or integrate reports into your existing applications.

For all .NET languages, the Report Designer remains the same, and the code used to control viewing reports and report engine calls will vary only slightly between languages, due to different syntax rules and conventions. For example, if you were binding a report to a Web Forms Viewer in VB .NET, the syntax would look something like this:

```
crystalReportViewer1.ReportSource = my_Report1
```

The same code can be ported to C#, with only one rather minor syntax change:

```
crystalReportViewer1.ReportSource = my_Report1;
```

So for developers who are creating applications with different languages or deployments, the concepts are the same, regardless of the language used.

Integration Methods

If you are new to Visual Studio .NET in general and have not used Crystal Reports .NET before, another thing you'll note is that the way that we integrate reports into both Windows and Web applications is different. Before Visual Studio .NET, Crystal Reports developers had a number of different integration methods they could choose from for Windows applications, such as an ActiveX control, Automation Server, or direct calls to the Crystal Reports Print Engine. For Web applications, Crystal Reports shipped its own Web component server and report viewers, allowing developers to integrate reporting into their applications.

Although the report integration solution provided for Windows development seemed to make most developers happy, the Web integration provided with Crystal Reports left something to be desired. There were inherent problems with configuration, scalability, and reliability, meaning that the Crystal Reports Web development platform could not be used to create scalable enterprise applications.

With the introduction of Visual Studio .NET, it was possible to bring both Windows and Web development into the same framework. The Crystal Report Engine is now a COM+ object wrapped around an updated version of the Crystal Reports Print Engine you may have worked with in the past. The Report Engine can be used to customize features at run time and also takes care of report processing.

When working with Crystal Reports for Visual Studio .NET, you have a choice of either leaving the report on the local machine (using that machine's resources to process and display the report results using the Windows Forms Viewer), publishing it to a Web server (using the Web Forms viewer), or publishing it as a Report Web Service that can be consumed and viewed by either the Windows or Web Forms Viewer.

Each of these integration methods will be covered in its own chapter, starting with Chapter 4, "Report Integration for Windows-Based Applications."

Ease of Use

Integrating a report into a Windows application is as simple as dragging the Crystal Report Viewer from the toolbar onto a Windows Forms Viewer and binding the viewer to a report file. (You could also create a report using the integrated designer.) The Crystal Report Viewer provides a feature-rich client for viewing reports, and at design or run time you can customize the appearance and options of the viewer, pass parameters, set properties, and so on.

For Web development, there is also the Web Forms Viewer, which communicates with the Report Engine (either on the local machine or on a remote server) to display a report page in DHTML format. This allows you to quickly integrate reporting into your Web applications; there are no runtime files required, and the report processing can be performed on the server.

Building Enterprise Applications

In addition to these enhancements, Crystal Decisions has also released Crystal Enterprise—a scalable, platform-independent report distribution, scheduling, and processing engine that can be used in conjunction with Crystal Reports and Crystal Reports .NET. It provides the back-end muscle to create applications that can support hundreds of users for both real-time and scheduled reports with a clustered, multi-server architecture that can span Windows, Linux, and Unix platforms.

Reports that have been published to the Crystal Enterprise framework can be accessed directly from within Visual Studio .NET and integrated into your application.

In addition to providing a scalable, multi-tier back end for reporting applications, Crystal Enterprise also has its own security layer (which can use Windows NT authentication, LDAP, and so on), internal structures (folders, objects, and rights), and scheduling engine, as well as distribution capabilities that can be used to build complex reporting applications without having to reinvent a solutions architecture just for reporting.

For example, if you needed to create an application that generates a report every week in PDF format and sends it as an e-mail attachment to 10 different users, you could create that functionality within your own application or you could use the inherent scheduling and distribution capabilities within Crystal Enterprise to make a handful of API calls to do this for you.

Crystal Enterprise includes .NET assemblies that give you quick access to all of the properties, methods, and events required to work with the Crystal Enterprise framework. Leveraging the functionality that is included by default with Crystal Enterprise, you can quickly create robust reporting applications in a fraction of the time it would take you to code these features by hand in your own custom application.

Another key area where Crystal Enterprise earns its money is with its clustering technology and multiple-server architecture; imagine in our earlier example that there are now 10 reports that go to 100 different people each day with a copy of the report and a link back to where they can view and search the live report.

The clustering within Crystal Enterprise ensures that these jobs get run regardless of what servers are up or down, and the distributed architecture means that you can add multiple servers to share the processing workload, including servers tasked specifically to run scheduled reports and process on-demand requests.

Although the cost of Crystal Enterprise may be off-putting to some developers, its integration with Crystal Reports .NET and its distributed architecture—which is beyond the scope of this book—will ensure that you have the scalability you need when your reporting application that serves 10 suddenly needs to serve 10,000.

Report Architecture

When you look at Crystal Reports .NET, one of the immediate differences between this version and previous incarnations of the product is its ability to create multi-tier reporting applications. In the past, most Windows applications used a two-tier approach with Crystal Reports, where reports ran on the local machine on which the application was installed.

With the introduction of Crystal Server for version 4.0 of Crystal Reports, a first attempt was made at developing a client-server version of Crystal Reports; but it wasn't until 1994, when Seagate Software acquired Crystal Reports and the corporate scheduling product Ashwin—which could be used to schedule programs processes and so on—was introduced, that multi-tier report applications became a reality.

The combination of the two products was first introduced in 1995 as Crystal Info and later renamed Seagate Info. Through the Seagate Info SDK, an additional processing tier was introduced to developers, with a server-based architecture that allowed reports to be run on a separate server and then returned to the client.

Chapter 1

Although the Seagate Info SDK seemed like a good idea, developers were slow to adopt the technology and looked for other ways to create multi-tiered applications.

This led Crystal Decisions to rethink their product roadmap, and using the basic technology and architecture from Seagate Info, they created Crystal Enterprise, which was initially released in 2001. Two of the core features of Crystal Enterprise were an open architecture and a powerful SDK that allowed developers to integrate Crystal Enterprise functionality (scheduling, multiple-servers, security, and so on) into their own applications. Since that initial release, Crystal Enterprise has been grown from strength-to-strength to become a robust, scalable platform for delivering Crystal Reports.

So for Visual Studio .NET developers, the introduction of Crystal Reports .NET provided a wealth of tools that could be used to build scalable applications, from simple applications integrating basic reporting, to functionality, to complex reporting applications that serve thousands of users. With the update for Visual Studio .NET 2003, Crystal Reports .NET provides an even more stable platform for a wide variety of reporting applications. These generally fall into one of the following two categories: single-tier and two-tier.

Single-Tier

Crystal Reports integrated with applications created in previous versions of Visual Basic were usually deployed as single-tier applications. In a single-tier application, a developer would use one of the various integration methods to combine Crystal Reports within their application and would then distribute the report file and all of the Crystal Reports .d11 and runtime files required to make the application work. When a report was run, it ran locally as a thick-client application, using the resources of the machine where the application was installed.

With Crystal Reports .NET, you can still create single-tier (sometimes called *fat*) applications and distribute the runtime files required to run and view a report. Some of the limitations found in applications created with previous versions of Visual Studio tools will still apply, including the need to redistribute the report file if any changes are required. A much better solution is to consider applications with two or more tiers.

Two-Tier

Most Web applications created with Crystal Reports .NET are considered two-tier applications, as shown in Figure 1-6. In the first tier, a Web application makes a request for a report and the report is processed on the Web server that hosts the application.

This architecture provides definite advantages over a single-tier application, including off-loading of the report processing and viewing to a server and a publish-once mentality for publishing a single copy of a report to a Web server that can be accessed by multiple users. However, with this type of two-tier architecture, your application will be limited by the number of users that can physically connect to a single Web server, and report processing will add a definite increase to this server's work load if used heavily for viewing reports.

Now, let's move on to an even better solution with an even thinner client.

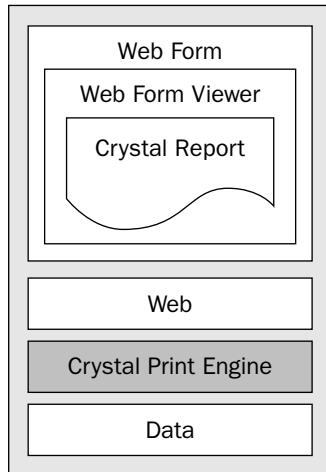


Figure 1-6

Three-Tier

A true three-tier reporting application, like the one shown in Figure 1-7, can be (but doesn't have to be) created using XML Report Web Services (covered in Chapter 6, "Creating XML Report Web Services"). A Report Web Service is a Crystal Report that has been exposed as a Web Service to be used (or consumed) by an application. Applications can connect to a Report Web Service, and the underlying report can be viewed using either the Web or Windows Report Viewer. This provides all of the functionality (view, drill-down, and export) found when integrating reports into a single or two-tier application, but with the report running on a server behind the scenes, the lightest client resources are required for actually viewing a report.

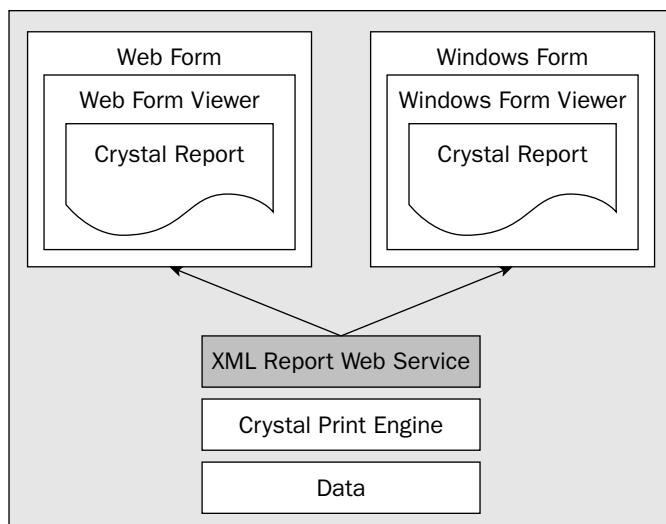


Figure 1-7

Chapter 1

In addition to being able to expose reports as Web Services for internal users, you can also publish Report Web Services to users external to your organization, providing a method for external users to access data held within your own data sources.

Multi-Tier Applications

When working with applications that are to be deployed to large numbers of users, you will probably want to consider moving to a multi-tier architecture, which is just a generalization of the three-tier concept and is shown in Figure 1-8, where components can be added as the application user base grows.

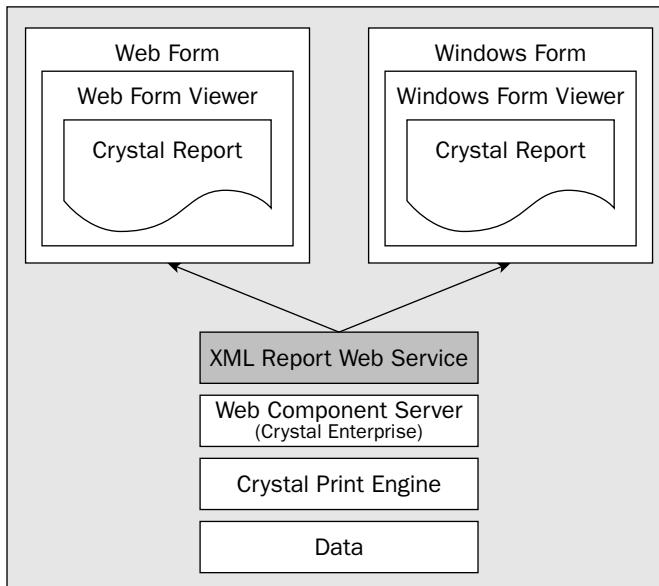


Figure 1-8

Crystal Enterprise is a Web-based, standalone solution for secure report delivery and distribution that can be integrated with Crystal Reports .NET. From within the Visual Studio .NET environment, you have access to the reports stored in the Crystal Enterprise framework and to a rich object model that exposes all of the Crystal Enterprise features and functionality (scheduling, security, and e-mail distribution) for use in your own application.

For more information on Crystal Enterprise, check out www.businessobjects.com/products/reporting/crystalenterprise

Report Designer

You can use the Crystal Reports Designer, shown in Figure 1-9, to create a report from scratch, or you can use a number of experts (similar to wizards) to help you get started. The interface is similar to the retail versions of Crystal Reports, and it shares enough similarity that existing report developers should have no problems making the transition to the .NET version. With that said, there are some specific options and features that are unique to this version.

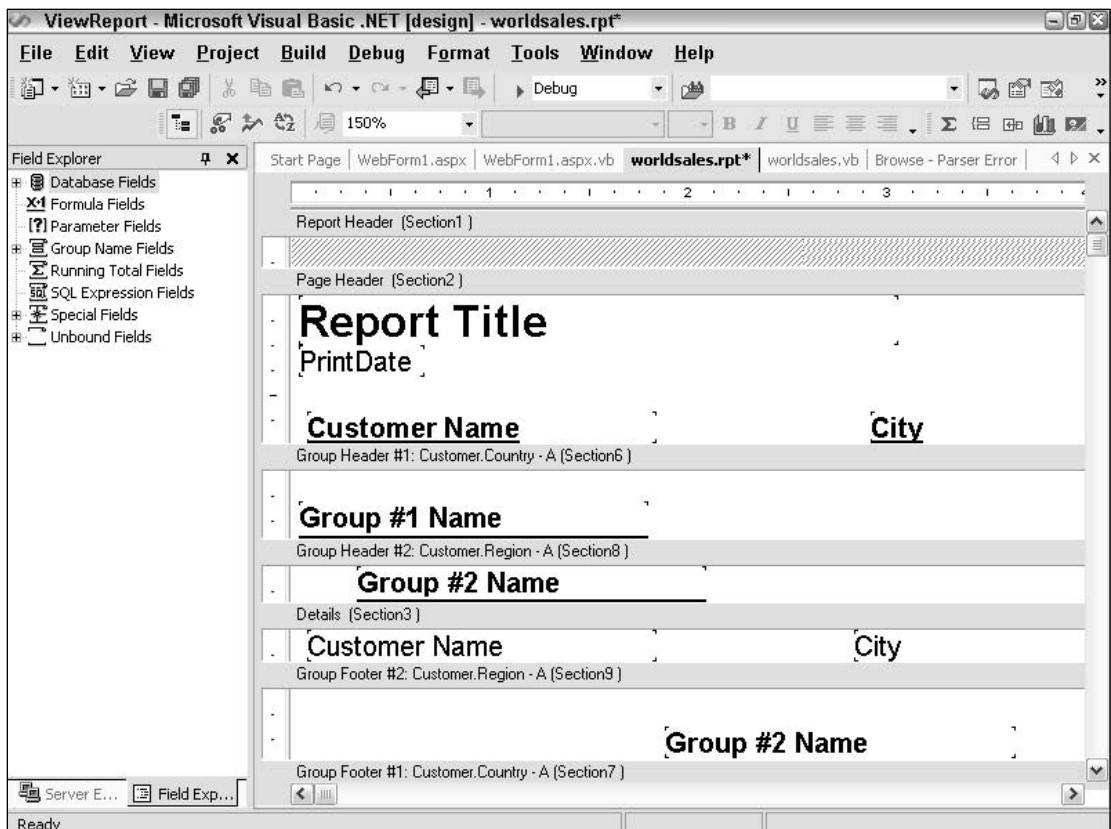


Figure 1-9

To start, Crystal Reports .NET has extended support for a number of data sources, including ADO .NET, OLE DB (ADO), ODBC (RDO), Access/Excel files (DAO), Crystal Field Definition files (from previous versions of Crystal Reports), and XML. When working with these data sources, Crystal Reports .NET can utilize either a *pull* or *push* mode of data retrieval.

To create a report that pulls the required data, you can create a report from a data source just as you normally would and let Crystal Reports handle writing the SQL statement, submitting the statement to the database, retrieving the records, formatting and displaying the records, and so on. This is the most common mode of integrating reports into applications and does not require any additional coding.

In push mode, a report can be created from a data source and used within your application, but it is the application itself that is handling the hard work of connecting to the database, populating an ADO .NET (or other) recordset and then pushing that recordset to the report. From that point, Crystal Reports will format and display the records it has received.

This method of integration requires more manual coding, but it provides more control over the dataset and report processing. Using the push mode to retrieve the data for your report means that you can use optimized SQL and stored procedures via ADO .NET to share database connections with other transactions that occur within your application, for example.

Incompatibilities

When using the Crystal Reports Designer available in Visual Studio.NET, you'll notice that there are a number of features that are available in the retail versions of Crystal Reports but are not supported here. A list of these features has been included for your reference:

- ❑ Geographic mapping is not supported in Crystal Reports for Visual Studio .NET. Map objects in Crystal Reports are implemented through third-party technology provided by MapInfo, and this has not yet been ported over to the .NET Report Designer. If you want to use existing reports that have maps with Crystal Reports for Visual Studio .NET, you can still do so, but the map objects will appear blank.
- ❑ OLAP (Online Analytical Data Processing) data sources and the grids that display OLAP information within a report are also not supported. If you are using an existing report that displays an OLAP grid, this area will be shown as a blank.
- ❑ Crystal Dictionaries, Crystal Queries, and Seagate Info Views are not supported. If you need to use an existing report that is based on any of these file formats, you will need to recreate the report directly from the database or data source itself.

Up until now we have looked only at previous versions of Crystal Reports. In August 2002, Crystal Decisions released Crystal Reports 9.0, which shares the same file format as Crystal Reports .NET and recently updated the product to version 10.0. It includes a standalone Report Designer, which does not require Visual Studio, as well as an updated Report Designer for use within the Visual Studio .NET environment. Thus you could have someone else create reports for your application without having to train them on how to use Visual Studio .NET.

Also included are new components for use with .NET—including increased data access, more productivity features, and a mobile viewer with associated tools that works with the .NET Mobile Internet Toolkit. For more information on Crystal Reports 10.0, visit www.businessobjects.com/products/reporting/crystalreports/.

Crystal Reports .NET Benefits

Now that we have looked at some of the differences among versions of Crystal Reports and at some of their uses and limitations, we need to have a look at some of the reasons you should be excited about this version and how your applications can benefit from the features we talked about earlier.

Leverage Existing Development and Skills

Crystal Reports can leverage the existing reports you have created, regardless of version. If you already have a suite of reports created in version 7.0, for example, you can quickly import them into Crystal Reports .NET, and they will be ready to be integrated in your application. In addition, the report design process remains the same, with a number of experts to guide you through report design and the same familiar design concepts, formula languages, and features you have used in previous versions. A word of warning: You can import reports from older versions of Crystal Reports .Net, but not the other way around; once you have opened or edited a report in Crystal Reports .NET, it uses a Unicode file format that is incompatible with previous versions.

Tight Visual Studio .NET Integration

From within Visual Studio, accessing a new report is as easy as selecting Project → Add New Item and then selecting Crystal Report. There is no need to open a separate application to design reports, and all of the reporting features are available to you, allowing you to programmatically control the look and feel of a report, how it is processed and viewed, and so on.

Windows and Web Report Viewers

For a feature-rich report viewing experience, Crystal Reports .NET includes a report viewer for Windows Forms, which has been built using the Windows Forms Classes and provides all of the functionality users have come to expect from Crystal Reports, including drill-down, search, exporting, and so on. In addition to a robust report viewer for Windows Forms, Crystal Reports .NET also includes a thin-client report viewing control for ASP .NET, providing most of the functionality found in the Windows viewer in a zero-client (meaning no client is downloaded or installed) DHTML environment, with no additional plug-in or viewer to download.

Easy Deployment

Crystal Reports .NET includes a number of merge modules to make creating setup projects easier. Instead of manually determining the required .dlls and other Crystal-related components, you can simply add one of the merge modules listed here to a setup project:

- Crystal_Managed2003.MSM

For installing the Crystal Reports .NET managed components, including:

- CrystalDecisions.CrystalReports.Engine.dll
- CrystalDecisions.Web.dll
- CrystalDecisions.Windows.Forms.dll

- Crystal_Database_Access2003.MSM

For installing all of the database drivers and all non-managed components (Charting, Export Formats)

- Crystal_Database_Access2003_enu.MSM

For installing select language-specific components

- REGWIZ.MSM

For tracking registration details and license keys

One of the most common errors when deploying a Crystal Reports .NET application is forgetting to change the LicenseKey property in your setup project. This must be set or an error involving keycodev2.dll may occur.

In addition to the merge modules listed earlier, you may need to include the VC_CRT and VC_STL modules if you are reporting from ADO recordsets, as the Crystal Reports database driver crldb_adoplus.dll relies on the files within these modules.

Chapter 1

For more information on deploying your Crystal Reports .NET application, go to Chapter 10, "Distributing Your Application."

ADO .NET

With the introduction of ADO .NET, data access has become much easier, and Crystal Reports .NET can take advantage of ADO and the ADO .NET dataset. Instead of having to work out how to access various data sources, Crystal Reports .NET can simply access the ADO .NET dataset as the source for any report you may create.

XML Report Web Services

For sharing reports and creating tiered applications, XML Report Web Services are invaluable. Within the Visual Studio IDE, you can create a Web Service from a report file with two clicks. From that point, Report Web Services can be exposed to users inside and outside of your organization and can be consumed using one of the new viewers included with the product. To optimize the report pages coming over the wire, XML is used to send the report a page at a time to either the Windows or Web Report Viewer, which makes reports viewed from Web Services quick and responsive.

Installing Crystal Reports .NET

Crystal Reports .NET ships as a component of Visual Studio .NET and can be installed from the common Visual Studio .NET setup utility. If you are installing Visual Studio .NET for the first time, you may need to complete the Windows Component Update shown in Figure 1-10 before you can begin. The setup utility will look at your current configuration and determine whether you need to update any files or applications. If required, setup will guide you through the update process.

After you have completed the component update, you can install Visual Studio .NET. The option to install Crystal Reports for Visual Studio .NET can be found under the Enterprise Development Tools options when selecting installation components. By default, when you select the Crystal Reports option, all of the related components will be installed as well, as shown in Figure 1-11:

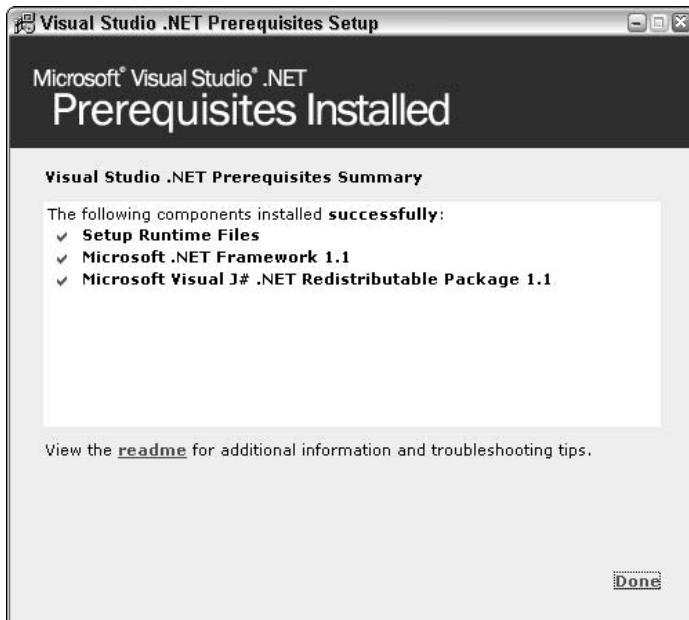


Figure 1-10

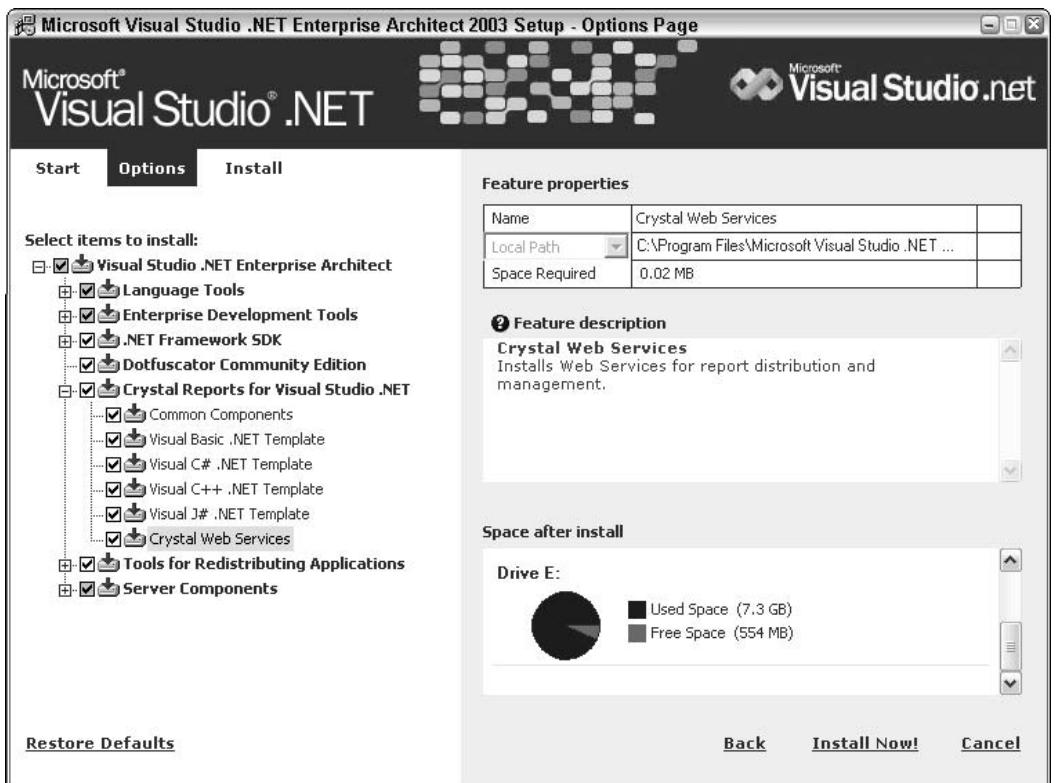


Figure 1-11

Chapter 1

Once it is installed, the Crystal Reports icon will appear on the Visual Studio .NET splash screen, and from the Add New Item menu, you should see the option to add a new Crystal Report, as shown in Figure 1-12.

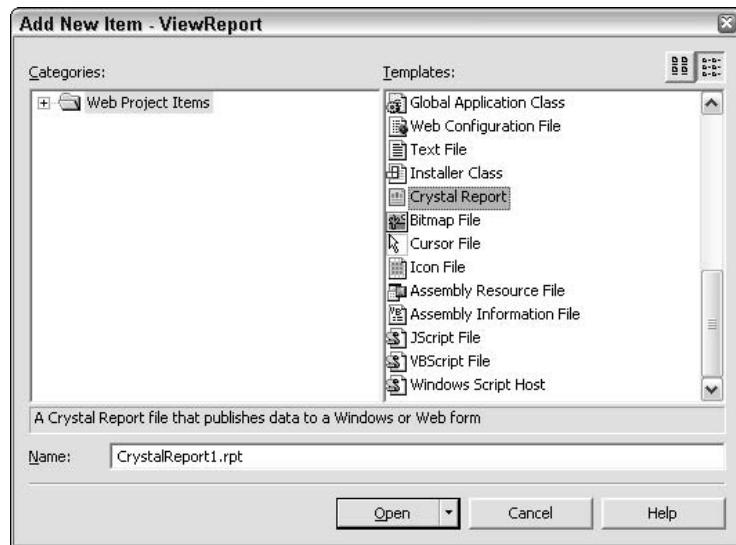


Figure 1-12

With the installation complete, we can jump right into looking at the samples that are installed with the product and learning about Crystal Reports .NET.

Learning from Sample Applications

Crystal Reports for Visual Studio .NET ships with a number of sample applications to help you get started. The majority of sample applications are simple, but each demonstrates some aspect of report integration, features, or functionality and provides a good learning resource if you are just starting out with Crystal Reports .NET or are new to this version.

Installing Sample Applications

The Crystal Reports .NET sample applications are installed by default and can be found in the Crystal Reports directory where you have installed Visual Studio .NET. These samples are in self-extracting files that you will need to run before you can open the samples within Visual Studio .NET. These

samples, shown in Figure 1-13, are located at `X:\Program Files\Microsoft Visual Studio .NET 2003\Crystal Reports\Samples\Code` (where `X:` is the drive where you have installed Visual Studio .NET).

There are two sets of sample applications available, both of which are bundled in self-extracting files—`WebForms.exe` and `Winforms.exe`. The first, which will extract to a folder marked `WebForms`, is written using ASP .NET and demonstrates the use of the Crystal Reports Web Forms Viewer. To view these samples, you will need to use IIS Internet Services Manager to create a virtual directory called `CRSamples` that points to the directory where you extracted the sample files. From that point, you should be able to access these samples from `http://localhost/CRSamples`.

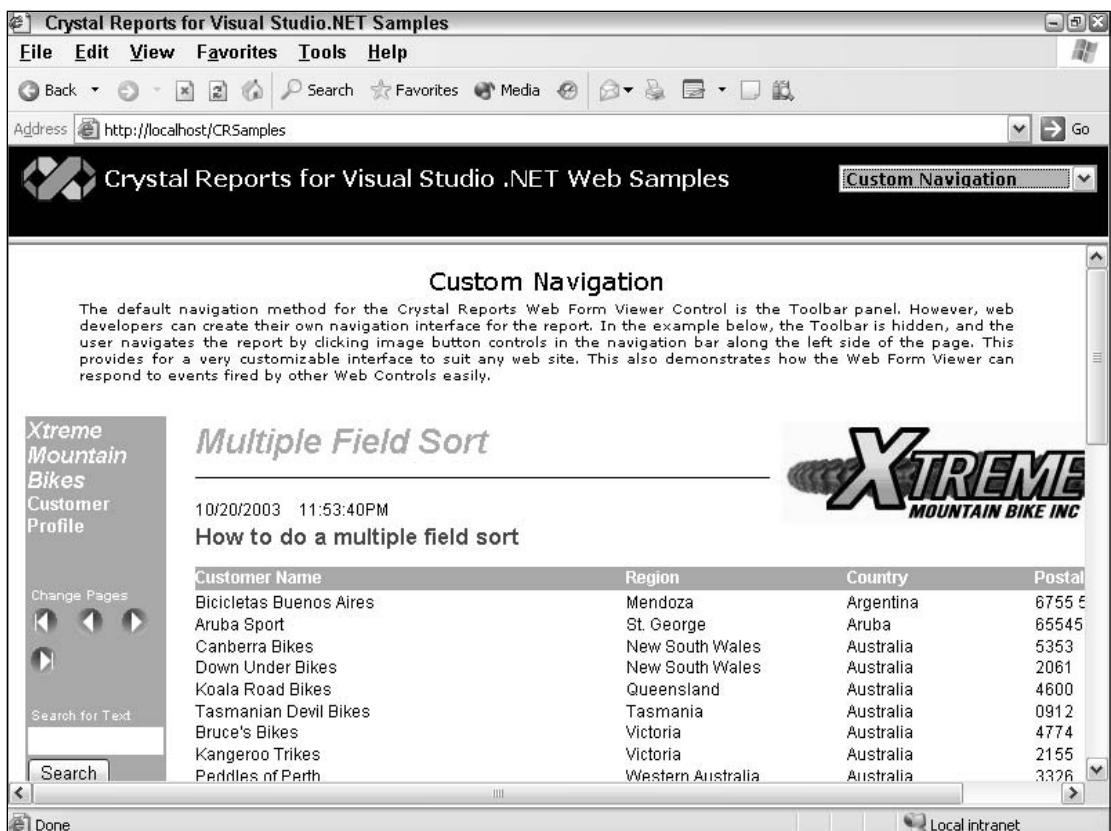


Figure 1-13

Chapter 1

The following samples are included in this set:

Sample	Description
Simple Page	The Simple Page sample application demonstrates using the Web Forms Viewer with one of the sample reports (World Sales). This sample demonstrates drilling down into the details of the report and some of the other features of the report viewer, including the report group tree, the viewer toolbar, and the page control options.
Custom Navigation	This sample demonstrates the amount of customization that can occur within the Web Forms Viewer. The viewer has a set toolbar by default, but developers who wish to control the look and feel of the entire page can control the toolbar or even create their own with minimal coding.
Interactivity	This shows how events can be fired when different areas of a report are clicked, which will change the text of the textbox in the upper right-hand corner. It also gives some insight into the events supported by the Web Forms Viewer.

There is also a [More Samples](#) link that will take you back to the Crystal Decisions Web site.

The second set of sample applications, for Windows Forms, is extracted from a self-extracting file and can be found in the WinForms folder. These samples demonstrate the use of Crystal Reports .NET with Windows applications and include separate projects for Visual Basic and Visual C#. Both of these projects demonstrate a simple Preview implementation of the Windows Forms Viewer and allow you to select a report to view. Once you have selected a report file (there are a couple located in the Reports directory of the Samples folder), the report is bound to the viewer; the Print Engine runs the report and uses the viewer to display the results.

Sample Reports

In addition to sample applications, there are also sample report files available for you to use in your testing and development. There are two different sets of reports available in the Reports directory of the Samples folder: Feature Examples demonstrate different features and functionality within Crystal Reports .NET (Charting, Embedded Hyperlinks, Sorting, and so on), and General Business reports are typical of reports that may be created and used in business (Income Statement and World Sales Report, for example).

All of these reports have been created using the sample Access database that ships with Crystal Reports .NET and are indispensable to use when debugging. If you are having difficulty integrating your report and can't determine whether it is your code, the viewer, or the report designer itself that is not working, you can substitute your report with one of the sample reports and at least eliminate one option!

Sample Data

A sample database has been included with Crystal Reports .NET, and the sample reports listed previously are based on this database. The Xtreme Mountain Bike Company database (*xtreme.mdb*) shown in Figure 1-14 is an Access database that contains tables for Customers, Orders, Products, Suppliers, and Employees. It does not require a copy of Access to be installed or loaded on to your machine to be used.

Tutorials

Crystal Decisions has its own Web site dedicated to developing reporting applications with Visual Studio .NET, and it includes a number of tutorials or walkthroughs that can be used to get up to speed quickly with the product. The Web site is located at www.businessobjects.com/products/dev_zone/. You will need to register before you can download the tutorials or other materials.

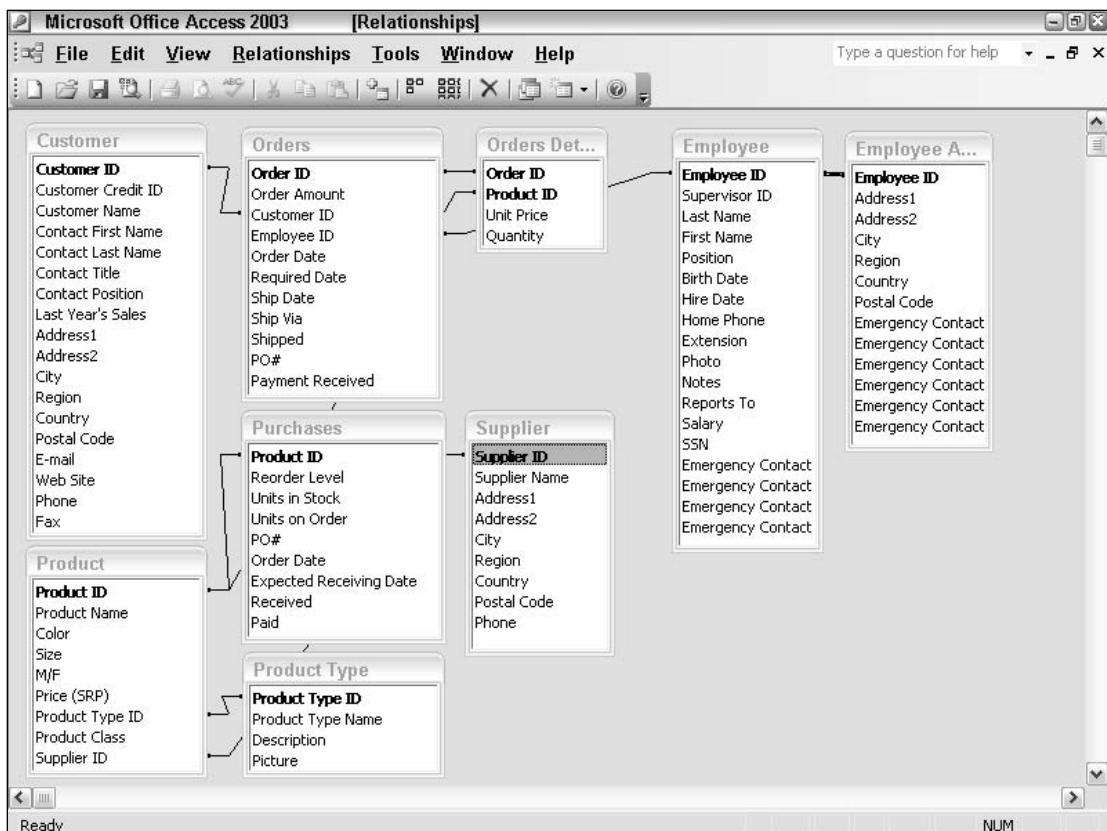


Figure 1-14

Chapter 1

There are a number of tutorials available:

- Reporting off ADO .NET Datasets
- Viewing a Report in a Web Application
- Designing and Viewing a Report in a Windows Application
- Exposing Reports as Web Services
- Interactivity and Reports in Web Applications

Most of these tutorials can be completed using the sample database and reports that ship with the product, or you can go through the tutorial using your own reports and data source.

You will also find a number of sample reports, applications, tutorials and walkthroughs on the Crystal Developers Journal Web site, available at www.crystaldevelopersjournal.com (shown in Figure 1-15). In addition to their own articles and content, the site serves as a clearing house with links to other Crystal-related content on the Web.

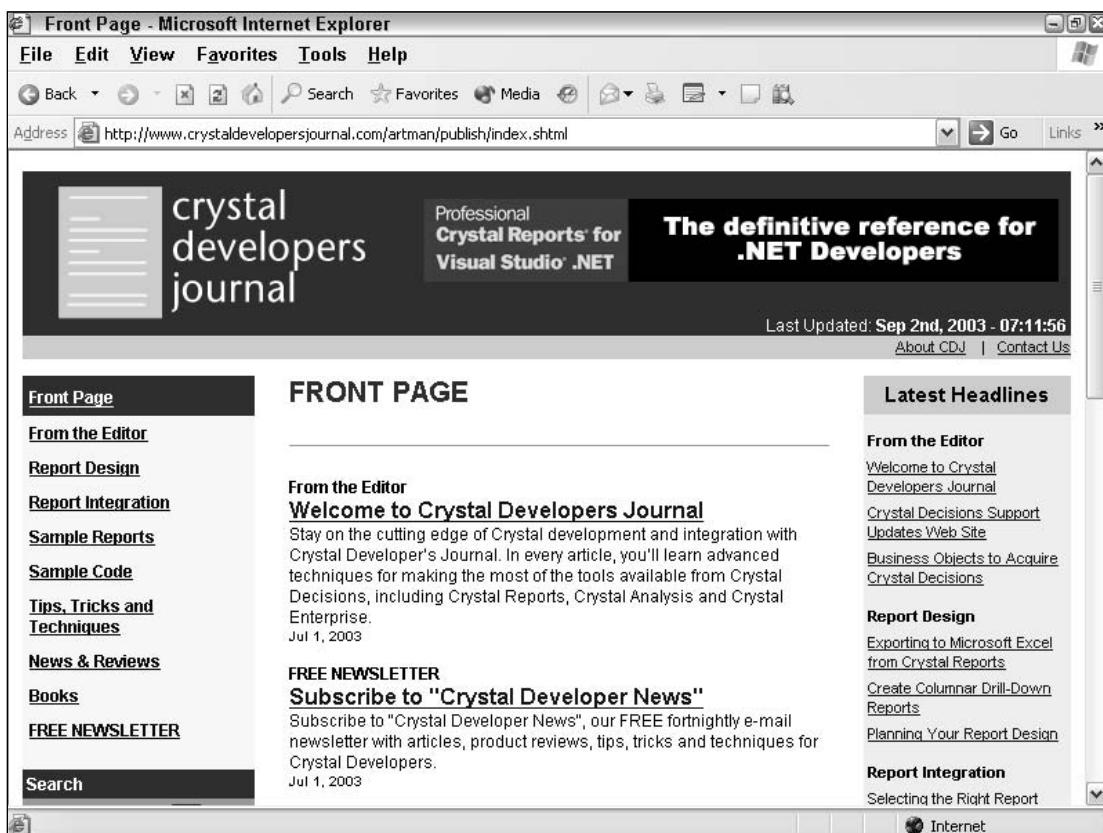


Figure 1-15

You can also find more information (and post a question if need be) on Microsoft's public newsgroups. The majority of Crystal-related questions, regardless of version or language, get posted to microsoft.public.vb.crystal, but there are always a few questions posted in the general dotnet newsgroup microsoft.public.dotnet.general.

You will need a newsgroup reader, such as Outlook Express, to access these newsgroups. You can also visit Microsoft's HTML version of the newsgroups on their Web site.

Finally, Crystal Decisions maintains its own Web-based forums at <http://community.businessobjects.com>, where you can post questions and get some answers. Crystal Decisions does not monitor these forums, but generally the advice is good, and you can always find someone who is willing to help. While you are on the site, make sure you register your copy of Crystal Reports .NET for free access to technical support and updates (and the requisite marketing e-mail or two).

Summary

Crystal Reports for Visual Studio .NET 2003 builds on the reporting technology found in Visual Studio .NET 2003 and is a powerful addition to the .NET toolset, designed to take advantage of the new .NET development framework. Using the Crystal Reports Designer, you can quickly create or modify reports without having to leave the Visual Studio IDE. When it is time to integrate your report into either a Windows or Web application, Crystal Reports includes a number of viewers that you can quickly integrate into your application. With a scalable back-end processing architecture, Crystal Reports for Visual Studio .NET should be the only tool you need to integrate reporting into your enterprise applications.

In the next chapter, we'll move beyond the product overview to actually start creating reports and integrating them into our own development.

2

Getting Started with Crystal Reports .NET

Now that you have a better understanding of Crystal Reports .NET's features and functionality, we can start to look at the tool itself and how it is used.

In this chapter, we will be looking at the Crystal Reports Designer within Visual Studio .NET and learning how to create and import reports for use in Windows or Web applications. This will include:

- Planning your report design
- Creating a report using an expert
- Working with the report design environment
- Report design basics

By the end of the chapter, you will have the skills to develop your own basic reports and navigate through both the user interface and the reports you have created. If you have used Crystal Reports before, some of the material in this chapter will be familiar. Crystal Reports .NET builds on the features and concepts found in previous versions of Crystal Reports, but there are some things that are unique to this version, which will be identified throughout this book.

Keep in mind that this chapter is not designed to be the exhaustive reference on report design; it is just to get you started. Chapter 3, "Designing Reports," is dedicated to all of the advanced report design concepts that you will need to create complex reports.

The Sample Files

In the C:\Crystal .NET2003\Chapter02\ folder you will find all of the sample reports, like the one shown in Figure 2-1 and other reports that illustrate similar concepts, as well as the application we build in this chapter:

- ❑ CustomerListing—a report that we will create with the Report Expert
- ❑ ViewerDemo—a small application that you can use to preview some more advanced sample reports

The sample reports utilize the Xtreme sample database that ships with Crystal Reports .NET. By default, this is located in at C:\Program Files\Microsoft Visual Studio .NET 2003\Crystal Reports\Samples\Database\Xtreme.mdb and will have an ODBC DSN that points to this location so Crystal Reports .NET can see the database. If for some reason, you don't have this DSN, you will need to create one using the ODBC Data Source Administrator found in the Control Panel under Administrative Tools (sometimes also called Data Sources in other operating systems) as shown in Figure 2-2.

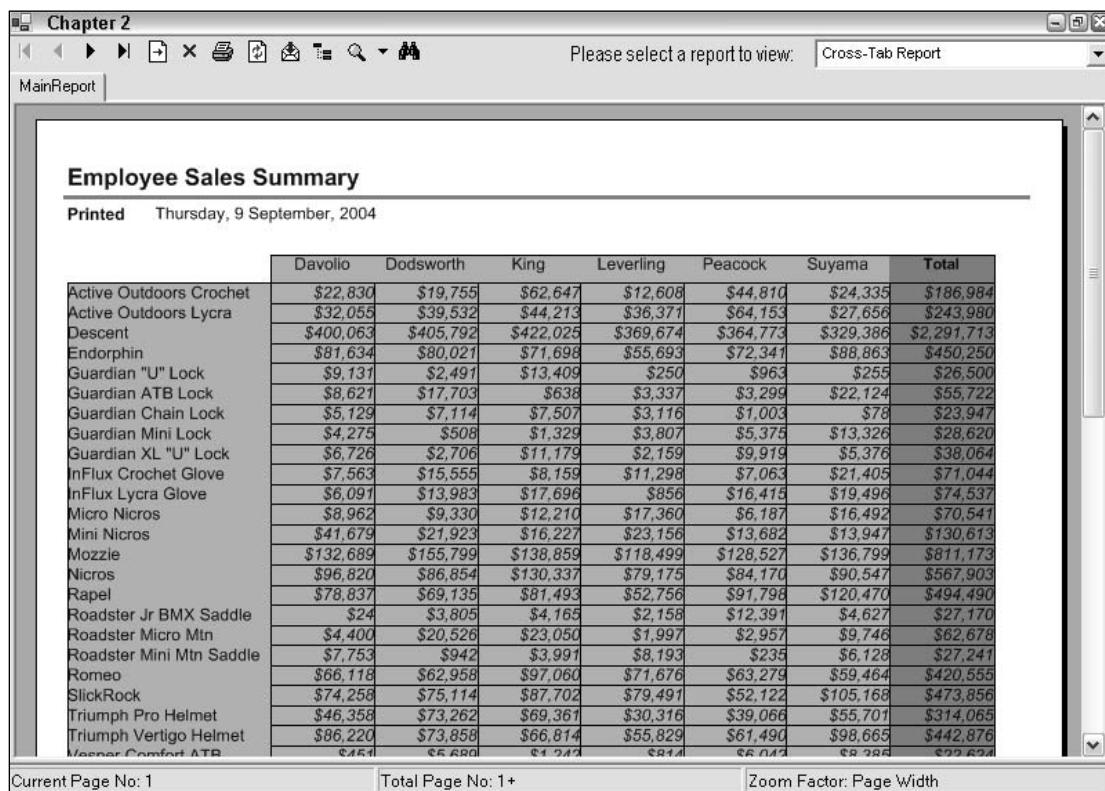


Figure 2-1

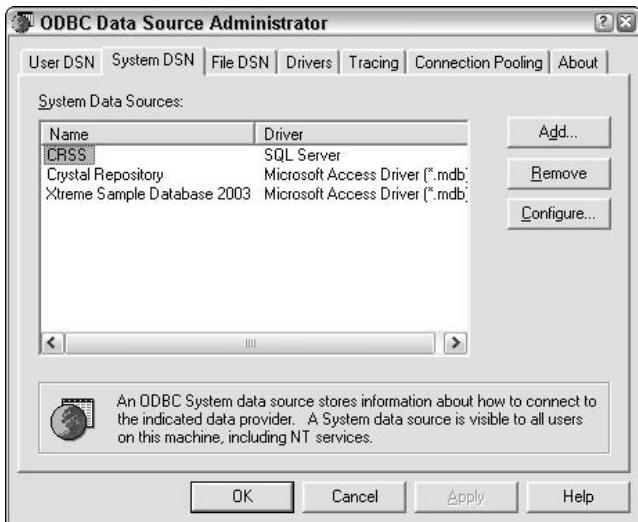


Figure 2-2

Select the System DSN tab and click the Add button. This displays a list of drivers for which you want to set up a data source. We are using a Microsoft Access database, so select Microsoft Access Driver (.mdb) and then use the next window to locate the database and give it a name (Xtreme Sample Database 2003), finally selecting OK to finish.

Planning Your Report Design

Before we actually get into the technical report design, we need to spend a little time planning our report to alleviate some of the problems we might face later when we actually sit down and develop the report. Often developers will create reports that they think the user will want, using all of the latest features, bells, and whistles, only to find that the end user would be happy with a simpler report that was easier to read and understand.

Another problem is that a developer will often sit down and crack open the Report Designer without really knowing what the end product will look like or considering how the end user will use what has been created. What is called for is a bit of planning before we jump right into the technical aspect of designing reports. By planning our report before we get started, we can deliver a report that meets the users' needs and expectations.

If you already have a report design methodology and it works for you, please have a read over this section anyway; you may find a different way of doing things or be able to incorporate some small part of it into what you already do. In any case, there are a number of reports associated with this section available in the downloadable code for this chapter. Feel free to modify these reports for your own use.

Chapter 2

In a very basic report design methodology, there are four steps to planning a new report:

- Gather report requirements
- Perform technical review
- Develop report prototype
- Design report

In most applications, reports are usually tied to a specific function or area of an application. For example, if you have created an application that is used for entering telephone sales orders from customers, chances are there will be a suite of reports tied to that function, showing sales summaries, order totals, and so on. The best way to determine what these reports should look like is to actually ask the people who will be using them on a daily basis.

If you are working in a large organization, you will probably have a business analyst who will interview the users, gather their requirements, and then communicate these requirements back to you. If you are in a smaller organization or, if like the rest of us, you are forced to take on a number of roles, you may gather these requirements yourself.

In either case, end-user interviews are the key to targeting a report's content. Organize the interviews with the actual end users (not their supervisors, personal assistants, or others), and ask them to bring along examples of reports they currently use or would like to use. This is your chance to find out what information they need to better perform their job. Be careful when interviewing, as users will sometimes come up with an arm-length wish list for reports they would like to see.

In the interview, the user should be able to tell you how the report is used and what decisions are made based on this information. If they can't tell you either of these things, either you are interviewing the wrong person or they really don't need the information they have asked for.

Once you have interviewed the end users, you should have a pretty good idea of what reports are required and how they will be used. From this point, there are many different ways of documenting the user's requirements (such as user requirements statements or use cases), but the most straightforward method is to create a formal Report Requirements document.

A Report Requirements document will outline what information needs to appear in the report, how the report is used (is it interactive or run in a batch?), and the general look and feel of the report. You may also create a mock-up of the report or a rough sketch of what it will look like. With your report requirements in hand, the next step in our method is to perform a technical review of the report's definition.

A good place to start with a technical review is to determine the data source for this report. Most likely the data source will be a relational database, but the data could also reside in spreadsheets, text files, OLAP (Online Analytical Processing) data structures, and even nonrelational data sources (like Exchange or Outlook folders). Once you have found the data source, you will need to dig a little deeper to determine the exact tables and views that can provide the data required. You may need to develop additional views or stored procedures to consolidate the data prior to developing a report (for speed and ease of use and reuse), and these will need to be documented, as well. Again, all of this information is added to your Report Requirements document.

For the next step of the technical review, you will need to investigate whether the design of the report is feasible. The user may request twenty columns when the page will fit only seven landscape, or they may have based the design on an existing report or spreadsheet created by hand. Once you are more experienced working with Crystal Reports, you will begin to understand how the tool works and the kind of output that can be achieved. In the meantime, browse through some of the sample reports that ship with the product to get a feel for the types of reports Crystal can produce.

Once you have completed the technical review, you understand where the data for the report resides, and you are comfortable that Crystal Reports .NET can deliver the required format, it is time to create a report prototype from your notes and preliminary sketches. This prototype can be created using Word, Excel, Visio, and other programs, but it should closely match the report's final layout and design. Again, another important check is to make sure that the layout and design you want can be created with the features and functionality that Crystal Reports .NET has available.

A good way to determine whether Crystal Reports .NET can deliver a particular format for data is to find a sample report that shows the data presented in the method you want to use. You also could create a small proof-of-concept report with sample data to test the design.

The prototype, combined with a formal Report Requirements document, will clearly communicate what the report should look like when it is finished. It also helps gain user acceptance for the design, as they can see what the finished product will look like even before you have opened the Report Designer. If you are working in a large team, this documentation will communicate the requirements to other report developers so you don't actually have to brief them on every single report that needs to be developed.

If you are working as a business analyst, application developer, and report developer all-in-one, it can also help you keep on track with the user's requirements and make gaining user acceptance that much easier. Once signed-off by the client, the Report Requirements is a contract, so should it turn out that the report designed isn't what the client wanted, you can point to this document and their signature on it!

Finally, with the documentation and prototype in hand, it is time to sit down, open the Report Designer, and design your report.

Creating a Crystal Report

In the next few pages, we are going to walk through the steps to create a customer listing report in an existing Windows application using the Standard Expert. You can open the solution and project that is available in the code download for this example (located by default at `C:\Crystal.NET2003\Chapter02\chapter2.sln`) or create your own.

When creating a report from scratch within Crystal Reports .NET, you have a number of experts that can be used to guide you through the report development process. In this section, we will look at how to use one of these experts to get started. Once you have finished working through the expert, you will have a report you can further develop using the integrated designer. Over time, as the interface becomes familiar, you may choose to start with a blank report and build it up piece-by-piece, but most developers use the expert at least to get them started.

To begin, create a new Visual Basic .NET project by opening Visual Studio .NET 2003 and selecting New Project. Select Windows Application from the dialog, call the project `CustomerListing`, and put it in the folder `C:\Crystal.NET2003\Chapter02`.

Adding a Report to Your Application

Adding a new report to your application is as simple as selecting Project → Add New Item and selecting Crystal Report from the list of available templates. Enter `customerlisting.rpt` into the name field and then click Open to insert the new report into your application. A separate tab will now appear for the report and allow you to use the integrated Report Designer to create your report, but before you can do this you will see the Crystal Report Gallery window, shown in Figure 2-3.

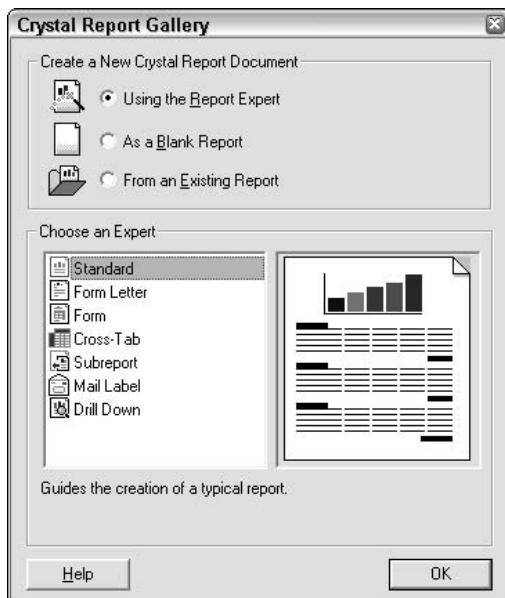


Figure 2-3

If you haven't registered Crystal Reports .NET, you will receive a nag screen asking you to register the product. This registration is specific to Crystal Reports and is required to be able to distribute your applications to other users. You can use the Registration Wizard that appears to register via the Internet, e-mail, fax, or mail and will receive a registration key that will eliminate the nag screen and allow you to distribute your application.

The first decision you will need to make about your report is to select how it will be created. In this instance, we are actually going to use one of the report experts to walk us through the report design, but there are other options as well, as shown in the following table.

Method	Description
Using the Report Expert	There are a number of different report experts available to walk you step-by-step through creating a report. This is the most popular method even for experienced report developers, as it provides a starting point for further report development.
As a Blank Report	This option is for experienced developers who are familiar with the Report Designer and want to build a report up piece by piece. This is often a good way to produce a report that is leaner and has fewer overheads built into it. When this is selected, the experts aren't available.
From an Existing Report	If you have an existing report that will serve as the basis for the report you wish to create, you can use this method to import your existing report. Again, when this is selected the experts aren't available.

Using a Report Expert

If you decide to use a report expert, there are seven different experts to choose from. All of the experts share some steps in common; in all of them you will be prompted for the data source for your report, for example, as well as for what fields will appear.

As we go through this chapter, we will be looking at all of the major steps involved in creating a report using the experts, but for now here is an overview of the different types of experts available:

- ❑ **Standard**—The Standard Expert is used most often and is the most generic. You can use the Standard Expert to create a column-based report that includes features such as grouping, sorting, and summaries. This expert has the ability to add charts, apply a number of predefined styles, filter records, and it also includes advanced analysis features like TopN, BottomN (for example, the top 10 or bottom 10 reports).
- ❑ **Form Letter**—By combining text objects and database fields, Crystal Reports .NET can be used to create form letters and statements. This expert guides you through creating a report and placing database fields and text within that report.
- ❑ **Form**—To support reports that are designed for a specific paper form, Crystal Reports .NET can use a scanned form or graphic as a guide for your report. Using the Form Expert, you will be able to underlay an image behind your report to correctly place fields on the form. From that point, you can either leave the image in place and print the form and fields on blank paper or delete the image and print the report directly onto your forms.
- ❑ **Cross-Tab**—Cross-tabs within Crystal Reports .NET look similar to a spreadsheet, with columns and rows of summarized data. Using the Cross-Tab Expert, you can create a report that will feature a cross-tab object in the report header.
- ❑ **Subreport**—Subreports are reports that are inserted into a main report. Subreports can be unrelated to the main report or you can pass parameters between the main report and subreports to determine the content to display.

- ❑ **Mail Label**—Crystal Reports .NET supports multi-column reports, and this functionality makes mailing labels possible. It includes a number of mailing labels predefined for common label stock from Avery and other suppliers (but only in Letter sizes), or you can create your own custom label size using this expert.
- ❑ **Drill Down**—The concept behind drill-down is that you display a summary in your report and users can drill-down into the summary to see the details that make it up.

For this chapter we will look at the Standard Expert, so click OK with Standard selected in the listbox in the Crystal Report Gallery window.

Selecting Your Data Source

The first step in any of the experts is to select your data source. There are seven different types of data sources we can use as the basis of our report, as shown in Figure 2-4.

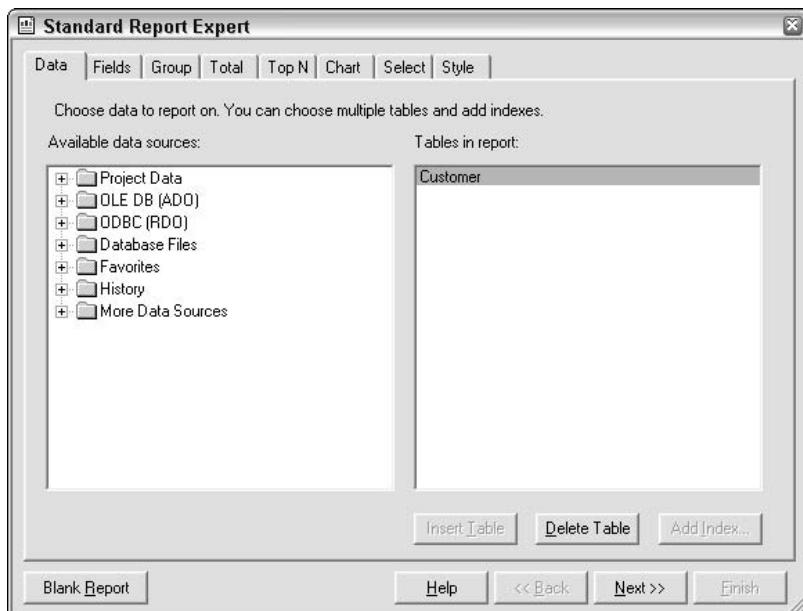


Figure 2-4

Which of these data sources you select as the basis of your report will depend on your own particular needs and the data sources available to you.

Data Source	Description
Project Data	Crystal Reports .NET can leverage the ADO .NET Framework and report directly from datasets that appear in your application. For more information on how Crystal Reports .NET can be used with ADO .NET data, please see Chapter 7, "Working with .NET Data."

Data Source	Description
OLE DB (ADO)	This folder is for data sources that can be accessed through OLE DB, including SQL Server, Oracle, and Microsoft Jet 3.51/4.00-accessible data sources (Access, Excel, Paradox, Dbase, and so on).
ODBC (RDO)	This folder is for data sources that can be accessed through an ODBC-compliant driver (which is just about every other data source). In addition to reporting from tables, views, stored procedures, and so on, Crystal Reports .NET will also allow you to enter an SQL command to serve as the basis for your report.
Database Files	This folder includes a number of file-type database formats, including Access, Excel, XML, and Crystal Field Definition files (TTX), used with previous versions of Crystal Reports and bound reporting.
More Data Sources	These include reporting directly from XML files, Access/Excel through DAO, and Crystal Field Definition Files (TTX).

When reporting from an ADO .NET data source, the underlying XML file (with the .XSD extension) is used to determine the details of the dataset, such as fields or lengths. Because this XML contains only the definition of the data and not the data itself, you won't be able to browse data from within the Report Designer when using this data source.

To select the table we are going to use in our customer listing report, double-click the node for ODBC (RDO), which will open the dialog shown in Figure 2-5.

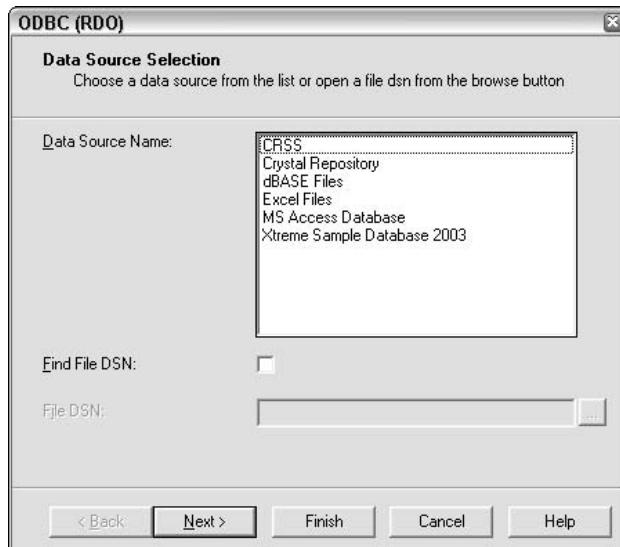


Figure 2-5

Chapter 2

Select the Xtreme Sample Database 2003 and click Finish. This data source should now appear in the Data tab of the Standard Report Expert, shown in Figure 2-6.

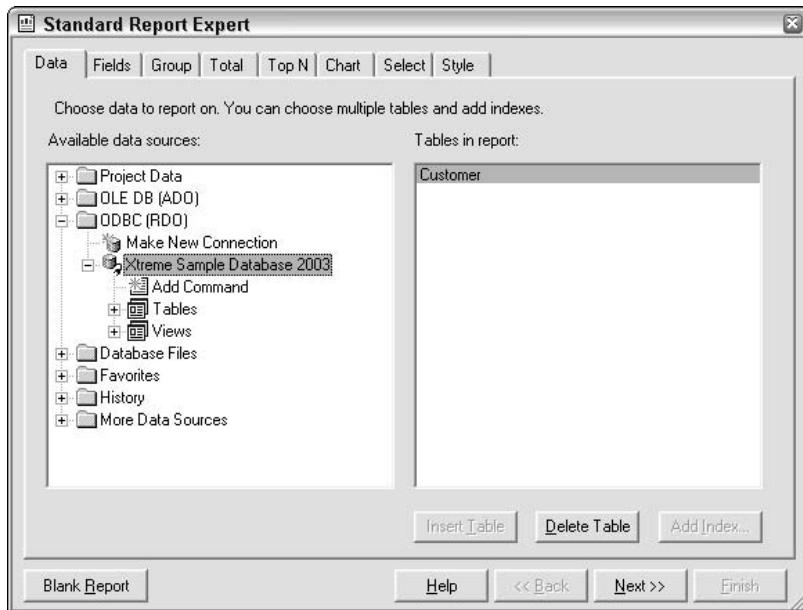


Figure 2-6

Click the plus icon next to Tables, locate the Customer table, and double-click it to insert this table into your report.

Multiple Tables

If you select multiple tables to appear in your report, a new tab will appear for Links, allowing you to specify how these tables or views are joined together. By default, Crystal Reports .NET will attempt to make these links for you, based on the primary keys, field lengths, and names. You can clear the links that have been provided by clicking the Clear Links button on the right-hand side of the Links dialog (or by highlighting each link and clicking the delete button), and you can redraw them by dragging one field on top of another.

By default, Crystal Reports .NET will join these tables with an Equal join, but you can change the join type by right-clicking on top of the link and selecting Link Options. If you have added a second table, delete it now because we will use only a single table in this example; to do this click the Data tab, select the second table you added from the Tables in Report column, and finally click the Delete Table button.

Choosing the Fields For Your Report

Now that you have selected the data source for your report, the next step in the Standard Expert is to select the fields that will appear in your report. To move on to this step, you can either click the Fields tab at the top of the expert or use the Next button at the bottom of the dialog.

To select a field, highlight the field and then use the arrows to move it from the left-hand list to the right, as shown in Figure 2-7. The fields will be listed using the notation TableName.FieldName, and if you are unsure of a field's contents or type, you can use the Browse Data button to browse the contents of the field. There is also a Find Field button for finding a particular field in long field lists, or you can click anywhere within the table and just start typing the name of the field; Crystal Reports will jump directly to the field.

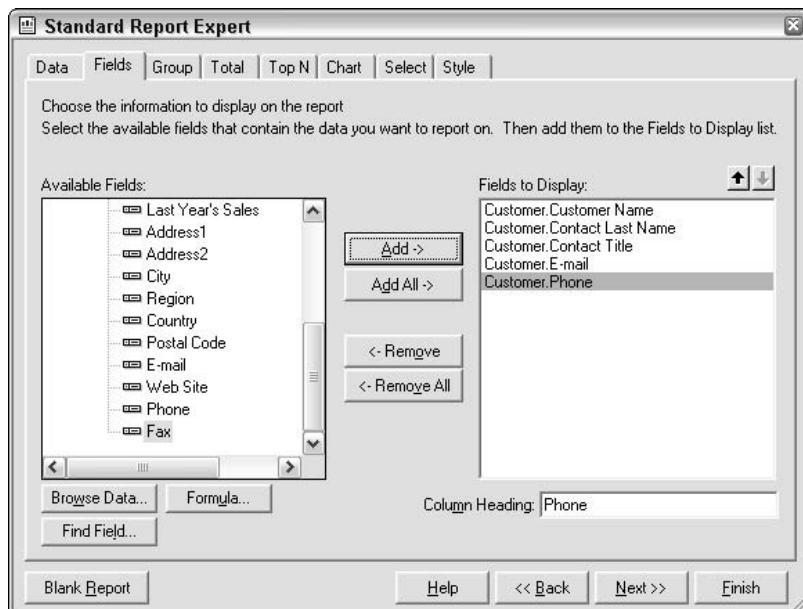


Figure 2-7

When you look at your report's design a little later, the fields you have selected here will be inserted into the report's detail section and a separate line will be printed for each record in the dataset. You can change the order these fields are inserted by using the up and down arrows at the top right corner of the dialog. They are inserted from left to right on your report, starting from the top of the list of fields that appear here.

For this report, we are inserting:

- Customer.Customer Name
- Customer.Contact Last Name
- Customer.Contact Title
- Customer.E-mail
- Customer.Phone

With each field, a corresponding field heading will be added to your report and placed in the page header. You can use the textbox marked Column Heading to change the default heading (usually the field name), or you can edit the headings later in the Report Designer.

Having selected the data source for your report and these five fields, you could click the Finish button to begin working with your report in the Report Designer, but we are going to push on and have a look at grouping and sorting the data.

Grouping and Sorting

After you have selected the fields that will appear on your report, you can choose which fields will be used for sorting and grouping. Click the Group tab and use the same concept as before—moving the field from the left-hand list to the right to select it for grouping.

This dialog can be confusing, as developers will select a group field and then preview their report, only to find that the data are sorted, but no group has been inserted. In order to create groups of records, you must specify a field in the dialog shown in Figure 2-8 and in the next dialog we discuss (the Total tab) and then select some summary to appear for each group.

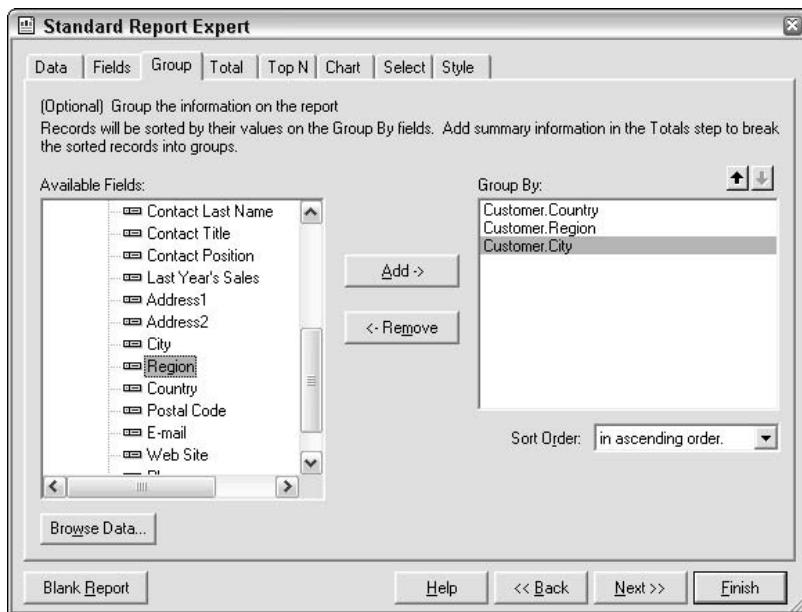


Figure 2-8

The fields we are selecting for the grouping in our report are:

- Customer.Country
- Customer.Region
- Customer.City

You can change the order of the fields you have selected by using the up and down arrows in the upper right corner of the dialog and then selecting a sort order from the drop-down list below.

There are four options available for sort order including:

Sort Order	Description
Ascending	For ordering the data from A-Z, 1-9, and so on.
Descending	For ordering the data from Z-A, 9-1, and so on.
Original	If your dataset is already sorted, this option will leave the data in its original sort order.
Specified	Used for creating your own custom groups and setting some criteria. Any records that meet the criteria would belong to the specified group.

If you select in a specified order, a second drop-down list and other items appear in the window, as shown in Figure 2-9.

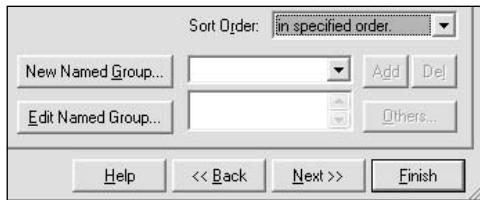


Figure 2-9

This is called *specified grouping*. It allows you to create named groups and specify some criteria for the group. For example, if you were working with a database that contained a Country field, you could create a new group by clicking on the New Named Group button, giving it the name of Asia Pacific, and then specifying the countries that are considered to be in the Asia Pacific group (Australia, New Zealand, and the Philippines, among others).

When setting up named groups, you are limited to the standard Crystal record-selection operators (equal to, not equal to, is one of, and so on.) and can only reference the field you are grouping.

You also have the option of dealing with the other records that fall outside of your grouping criteria by clicking on the Others box in the Standard Expert window. This will open the dialog shown in Figure 2-10 and will allow you to select what to do with the other, unused records.

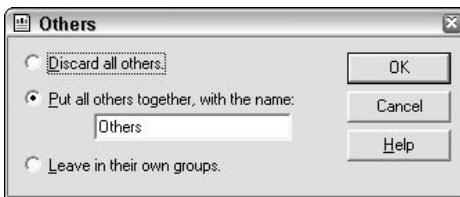


Figure 2-10

Although specified grouping is very useful for reordering records, it can be monotonous if you have to use it continually to create the same specified groups in multiple reports. The best practice for specified grouping is to use it only when necessary. If you find you are creating the same groups over and over again, you may want to consider putting this logic into a stored procedure or creating another lookup table in your database that will determine how your grouping is broken up.

For this example, we are not going to use specified grouping, so with your group set as Customer.Country, Customer.Region, and Customer.City in ascending order, the next step is to insert the summaries you want to see within your report.

Working with Simple Summaries

Click the Total tab in the Standard Expert window to add summary information to the report. Summary fields are calculated fields within a Crystal report that do not require a formula to be written. Encapsulating the most popular requests for calculations, like sums and averages, summary fields can be used to rapidly develop reports without a lot of repetitive coding.

To insert a summary field, you will need to select a database field (in this case, Customer.Last Year's Sales) and then choose a summary type from the drop-down list shown in Figure 2-11. We have chosen sum and have also checked the Add Grand Totals box at the bottom of the dialog.

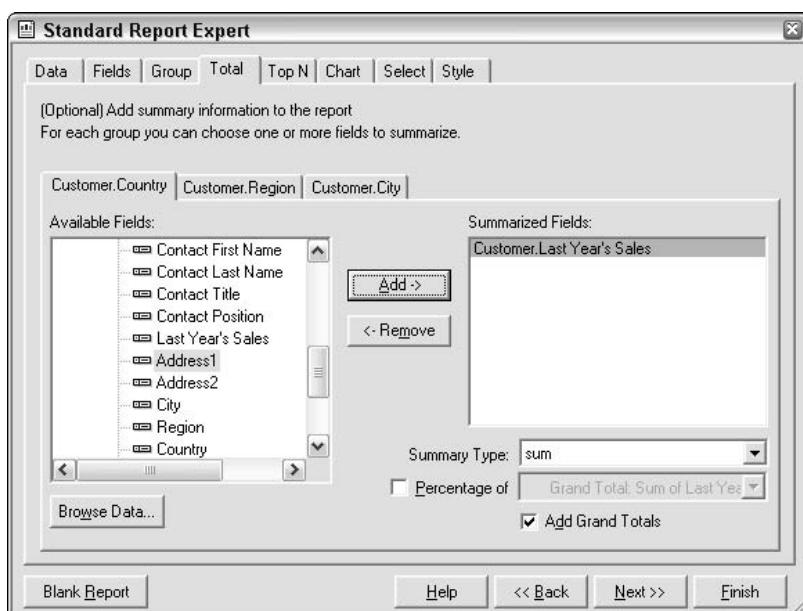


Figure 2-11

There are over 20 summary operators available for use, but unfortunately the documentation that ships with Crystal Reports .NET does not include a list of them or how they can be used, so a list of the most popular has been included here. Keep in mind that the summary fields available depend on the type of field you have selected; you can't calculate the average of a string field, for example.

Summary Function	Description
Sum	Calculates a sum of all items.
Average	Calculates the standard unweighted average for all items.
Maximum	Returns the maximum value.
Minimum	Returns the minimum value.
Count	Calculates the total number of all items.
Sample Variance	Calculates the statistical sample variance of a specific sample of items.
Sample Standard Deviation	Calculates the statistical standard deviation of a specific sample of items.
Population Variance	Calculates the statistical variance of the entire population of items.
Population Standard Deviation	Calculates the statistical standard deviation of the entire population of items.
Distinct Count	Calculates the number of distinct items. For instance, if two different instances of "David" appeared in the list, they would be counted only once.
Correlation	Calculates a measure of the relation between two or more items
Covariance	Calculates a measure of the variance between two or more items.
Weighted Average	Calculates an average, weighted by the number of times an item appears.
Median	Calculates the statistical median of all items.
Nth Percentile	Calculates the Nth percentile, where N is predefined number.
Nth Largest	Returns the Nth largest item, where N is predefined number.
Nth Smallest	Returns the Nth smallest item, where N is predefined number.
Mode	Calculates the statistical mode of all items.
Nth Most Frequent	Returns the Nth most frequent item, where N is a predefined number.

The corresponding functions to these summary fields are also available in the formula language, so you can also use them in complex formulas with branching and control structures if required. Keep in mind that their use as a summary field will be limited only to a few options or parameters passed to each.

Chapter 2

Using Analysis Features

In addition to simple summaries (which we have just looked at) and formulas (which are coming up in Chapter 8, “Formulas and Logic”), we also have the ability to add a number of analysis features to our report to help highlight information that may be important or otherwise missed. On the TopN tab of the report expert, we have five options for adding a bit of analysis to our report.

Analysis Type	Description
TopN	Orders your report groups according to a summary field, where you enter a number (N) and are presented with the TopN groups in order from the largest to smallest (for instance, you could create a top 10 report based on last year’s sales, to show your best customers). You can also use the options presented to discard the other records or place them in their own group.
BottomN	Will order your report groups according to a summary field, where you enter a number (N) and are presented with the BottomN groups (for instance, you could create a bottom 10 report, based on last year’s sales, to show your worst customers). Similar to TopN, you can use the options presented to group or discard the other records.
Sort All	Will order your report groups according to a summary field, either ascending or descending or by a top or bottom percentage, depending on the options you set.
Top Percentage	Will order your report groups according to a summary field, where you enter a percentage (N) and are presented with the Top Percentage of groups (for instance, you could create a Top 10 Percent report, based on last year’s sales, to show the customers in the top 10 percentile). Similar to TopN, you can use the options presented to group or discard the other records.
Bottom Percentage	Will order your report groups according to a summary field, where you enter a percentage (N) and are presented with the Bottom Percentage of groups (for instance, you could create a Bottom 10 Percent report, based on last year’s sales, to show the customers in the lowest 10 percentile). Similar to BottomN, you can use the options presented to group or discard the other records.

Keep in mind that all of these analysis options will be applied throughout your report and will apply to any graphs or charts you might insert in the next step of the report expert.

Charting and Graphing

For charting and graphing functionality, Crystal Reports .NET relies on a graphing engine created by 3-D graphics. In the report experts as well as the designer itself, you can add a number of different graph types to your report through the Chart tab, shown in Figure 2-12.

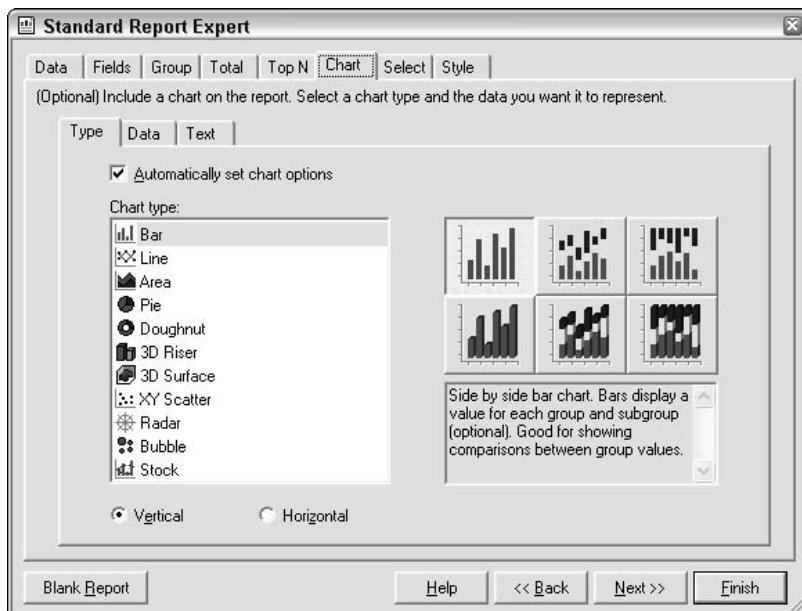


Figure 2-12

To create a chart for your report, select the Chart tab, select a chart type, and then select where the data will come from by clicking on the Data tab within the dialog. The most common type of chart is a Group Chart, which requires that both a group and a summary field be inserted into your report. If you would like more information on the different types of graphs, check out Chapter 3, "Designing Reports."

Filtering Your Report

When reporting from a number of different tables, the chances are you don't want to see all of the data in your report. Crystal Reports .NET follows the tradition set in previous versions of the product and has its own *Record Selection Formula* that dictates what records are returned to the report, so click the Select tab to create the selection formula for your report, as shown in Figure 2-13.

The Record Selection Formula is written using Crystal Reports .NET's own proprietary formula language, which in turn is translated to standard SQL and submitted to the database. When there is a feature that can't be translated to SQL, Crystal retrieves all of the records and uses its own Report Engine to apply the formula and filter the records.

When working with the Select tab in the report expert, the options are identical to those you will find in the Select Expert, which is used inside the Report Designer for record selection.

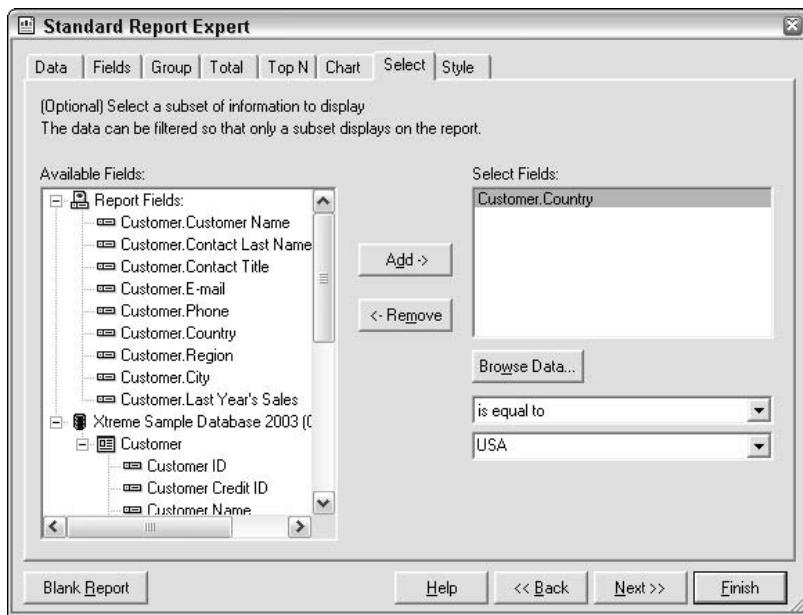


Figure 2-13

There are a number of basic operators available, depending on the type of field you are working with, including:

- Is equal to
- Is not equal to
- Is less than
- Is less than or equal to
- Is greater than
- Is greater than or equal to
- Is one of – (for building a list of items, such as Is one of Australia, UK, New Zealand; similar to the In operator in SQL)
- Is between
- Is not between
- Is like – (for wildcard searches where an asterisk represents many characters—for instance, *Zealand—and a question mark represents single characters—????Zealand)
- Is starting with – (for strings that start with a phrase entered)

Remember that this record selection is written to a formula and then translated into the SQL statement. We will look at some more advanced record selection a little later in this chapter and again in Chapter 8,

which deals with formulas and logic. For our purposes, we are not going to set any record selection on the report we are creating; we want all of the records to be returned from the sample database, so make sure nothing is in the Select Fields, and move on to the next tab, Style.

Selecting a Report Style

Finally, the last step of the report expert involves selecting a particular style for your report and adding a title using the dialog shown in Figure 2-14. There are 10 different predefined styles to select from and, no, you cannot add your own style to the list! You can also add a report title, in this case Customer Listing Report, which will be stored in the report file's Summary Information. A Report Title field will be added to your report's design. Add a title of Customer Listing Report for this report.

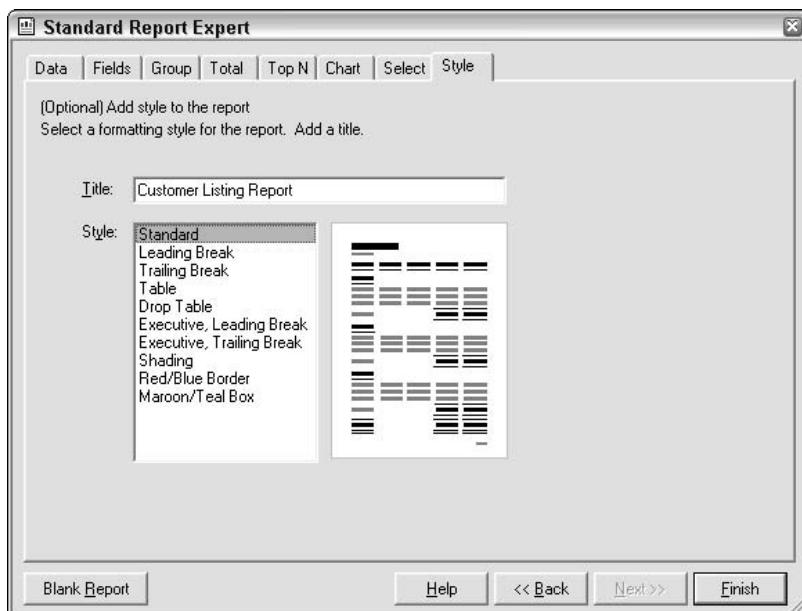


Figure 2-14

Adding Your Report to a Form

The final step in the Standard Expert is to click the Finish button, which will open your report inside the Report Designer within Visual Studio .NET.

Make sure that you get into the habit of immediately saving your report by clicking the Save or Save All icons within the Visual Studio .NET IDE. Although the Report Designer does have an undo feature, it is always nice to have a saved copy in case anything should go wrong.

Chapter 2

At this point, you probably want to have a look at how your report will appear when it is printed. Running the report will open only a blank form, though; the designer unfortunately does not include an integrated preview, so to preview your report, you will need to add the Crystal Reports Viewer to a form and then specify the report source to be the report you have just created.

Open `Form1.vb` in design mode. If you want, you can select the form and then change the text in the properties window to Customer Listing Report, so the user can then see what the report is about if they view the report on a computer. Finally, change the size field so it reads 700,500.

From the Windows Forms section of the Toolbox, drag and drop the `CrystalReportViewer` onto the form. Without this, any reports cannot be seen. If you can't see the Toolbox, click `View → Toolbox` to make it appear (the keyboard shortcut is `Ctrl+Alt+X`). Your form should now look something like the form shown in Figure 2-15.

Position the Crystal Report Viewer in the top left corner of the form, and drag its bottom right corner diagonally down, resizing the viewer to fill the whole form. Now when your report is displayed, it is nice and clear. Next, change the anchor property to Top, Bottom, Left, Right so if the form is resized, the Crystal Report Viewer, and more importantly your report inside it, will be resized to match.

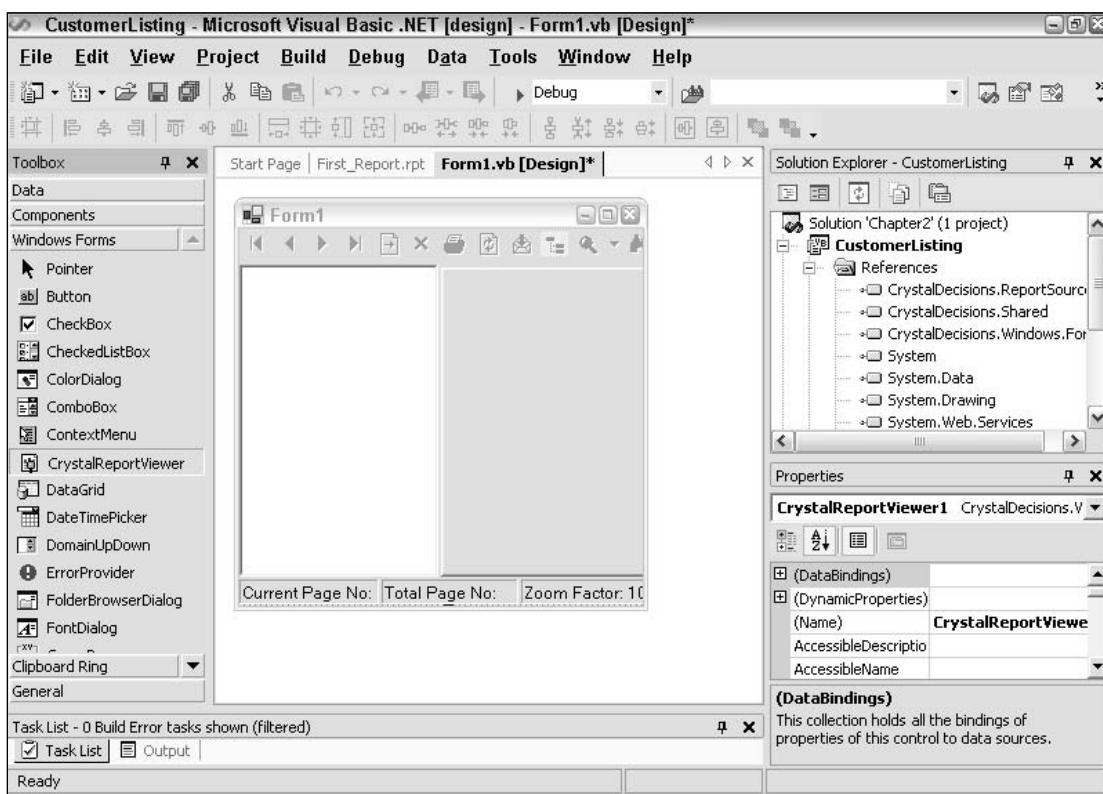


Figure 2-15

Getting Started with Crystal Reports .NET

From the Components section of the Toolbox, drag and drop the ReportDocument icon onto the form. This opens up a dialog like the one shown in Figure 2-16, from which you can choose the report to be opened in the form. Select the report we have just created, displayed as CustomerListing.customerlisting, and click OK.

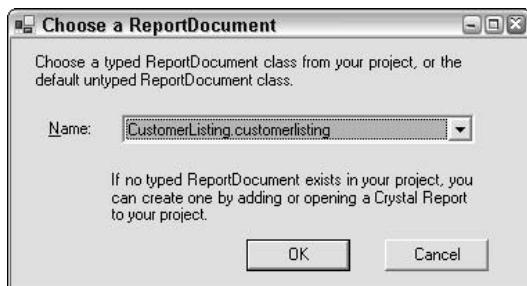


Figure 2-16

The customerlisting.rpt file is now shown in a shaded area at the bottom of the form designer, as shown in Figure 2-17. This shows that the report has been added, and many more reports can be too, using the same method we used here. However, for this example we want to display just the report we have just created, so we will move on.

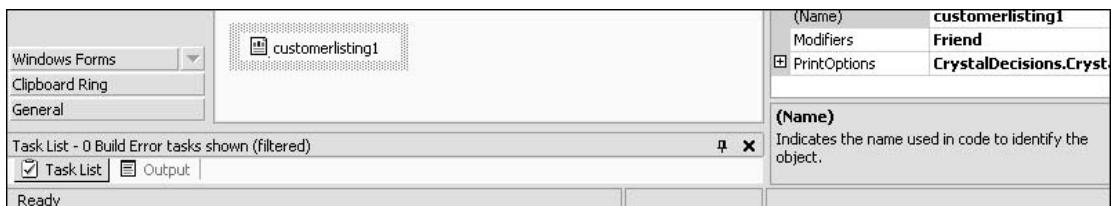


Figure 2-17

Finally, we want the report to open in the form when it loads, so double-click the form to open the code designer. The load procedure will be created because of this action, so all you need to do is tell the form where to find the report. Add the shaded line in the following example to your code:

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    ...
    'Windows Form Designer Generated Code
    ...
    Private Sub CrystalReportViewer1_Load(ByVal sender As System.Object,
                                         ByVal e As System.EventArgs)
        Handles CrystalReportViewer1.Load
        CrystalReportViewer1.ReportSource = New customerlisting()
    End Sub
End Class
```

That's it! Your basic sales report is complete, and all you have to do is click the small blue start icon in Visual Studio .NET to see it in action. Figure 2-18 shows the resulting report.

Chapter 2

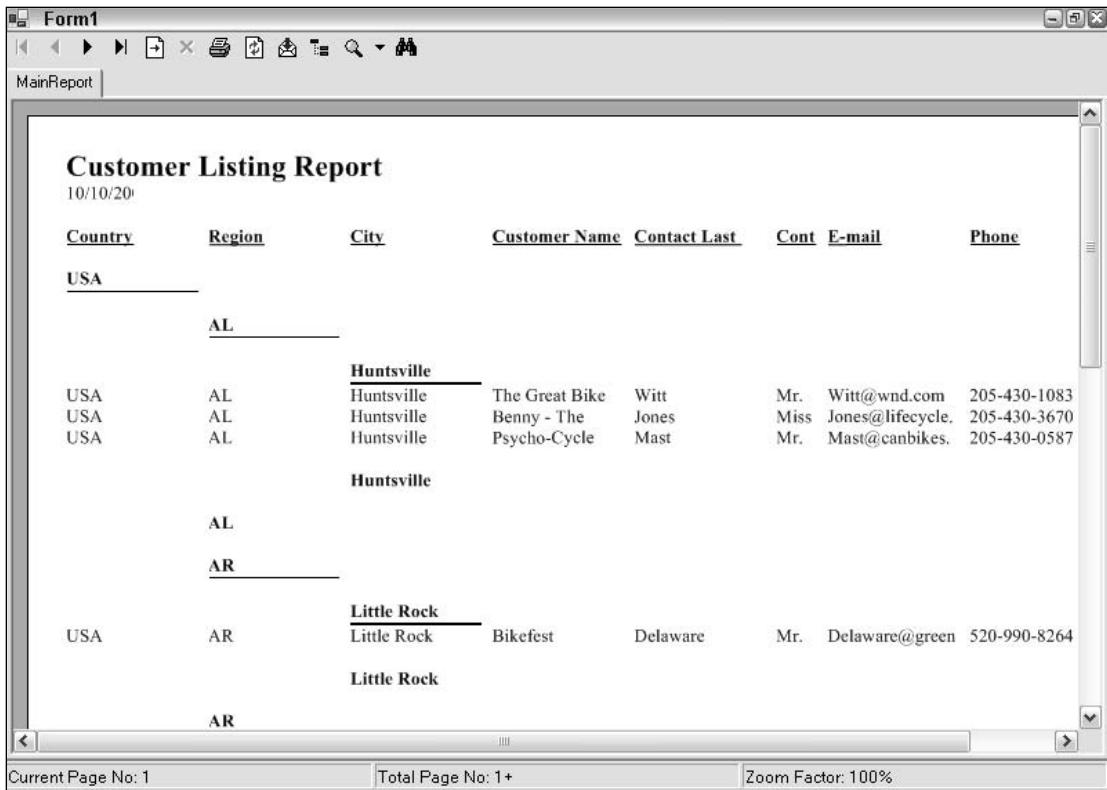


Figure 2-18

Because we added some grouping within the report expert, the report is grouped by country, region, and city. Select one of these from the tree view on the left of the window to jump to that information. As you can see from this example, Crystal Reports .NET can present a huge amount of data in a clear and organized manner.

Now that you have stepped through the many stages of the report expert, your report should match the `CustomerListing.rpt` that is available in the code download. With a basic report using an expert out of the way, it is time to take a look at the Report Designer itself. Make sure you save your project, as we will use this report as the foundation for some later examples.

Working with the Report Design Environment

Regardless of how many times you go through the report experts to create reports, you will spend most of your time working with reports in the integrated Report Designer within Visual Studio .NET, shown in Figure 2-19. If you have worked with a previous version of Crystal Reports, this interface will seem familiar, but there are subtle differences you'll notice when looking for a specific function or feature.

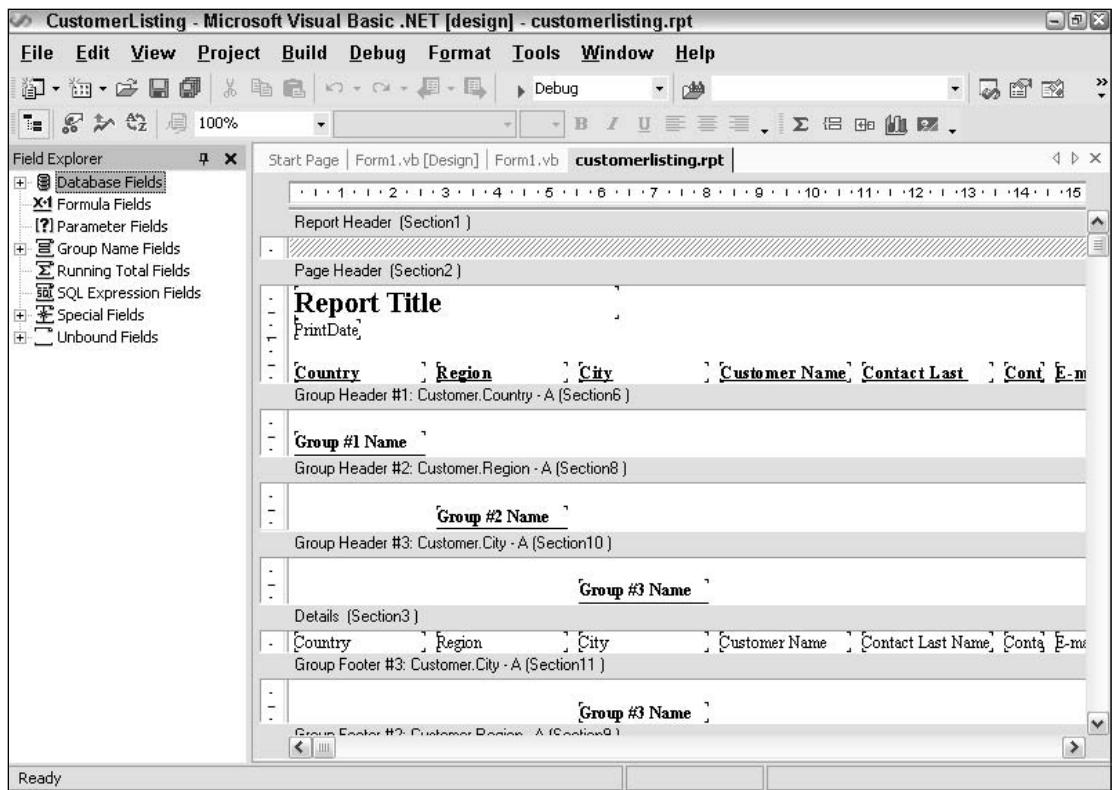


Figure 2-19

To understand how to work with the designer, we need to take a look at some of the components that make up the Report Designer.

Menus and Toolbars

To start, your familiar Crystal Reports menus and toolbars are gone; the constraint of being embedded in the Visual Studio .NET Framework means that we can't have exclusive control over the menus that appear in the IDE.

Instead, the majority of these options can be found in the toolbars, explorers, or right-click menus within the Report Designer. There are two toolbars available with Crystal Reports .NET, shown in Figure 2-20, one called Crystal Reports—Main and the other called Crystal Reports—Insert. The Main toolbar contains the formatting controls, such as font and size, and the Insert toolbar provides the facility to insert summaries, charts, and groups.



Figure 2-20

Chapter 2

For inserting fields into your report, there is a Field Explorer (which used to be invoked by selecting Insert → Field Object in older versions of Crystal Reports). You can drag fields directly from the Field Explorer onto your report.

I mentioned it before, but it is worth repeating here: if you close the Field Explorer, you will need to go to View → Other Windows → Document Outline or press Ctrl-Alt-T to view it again.

Setting Default Properties

By default, Crystal Reports .NET will have a number of properties preset. These include the font and formatting for fields in your report, the page size, margins and layout, and so forth.

For more control over the reports you create, you can actually change these default properties. One of the most common scenarios is that Crystal Reports .NET defaults the font to Times New Roman and sets a specific font size for different types of fields. Your standard report template may be in Arial, so you can either spend a considerable amount of time changing the font sizes for the different elements in every report you create, or you can set the defaults and be done with it once and for all; any reports you create from this point onwards will use these settings.

There are two sets of properties associated with Crystal Reports .NET. The first comprises the default settings, which are written to the local registry and exist for all reports created using that particular machine. (There is no easy way to port these settings between two machines unless you get up to some registry wizardry.) These options can be found by right-clicking in the Report Designer and selecting Designer → Default Settings. Through the default settings shown in Figure 2-21, you can control field formatting and fonts, database options, and a variety of miscellaneous options grouped together under the different categories.

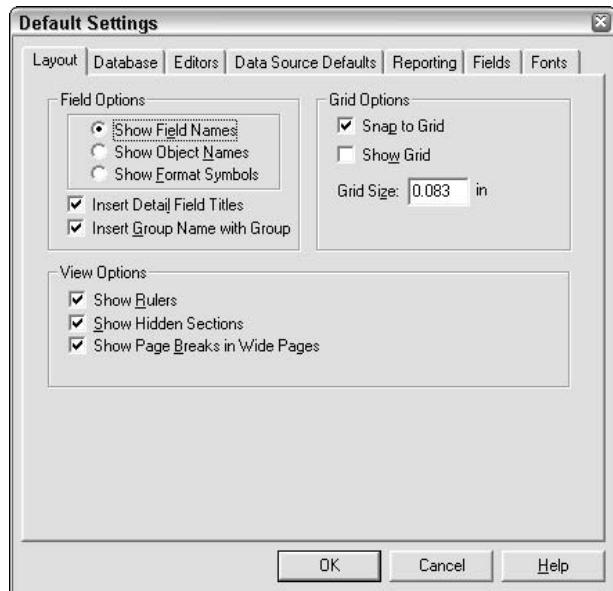


Figure 2-21

The second type of properties is the report options, which are specific to the report you are working with. To access the report options, right-click in the Report Designer and select Report → Report Options. There are fewer options in this dialog, and they are applied only to the report you are working on. Like the default settings, there is not an easy way to share these attributes with other reports. You will find options here for controlling how date-time fields are represented, for using indexes, and for sorting, among others, as shown in Figure 2-22.

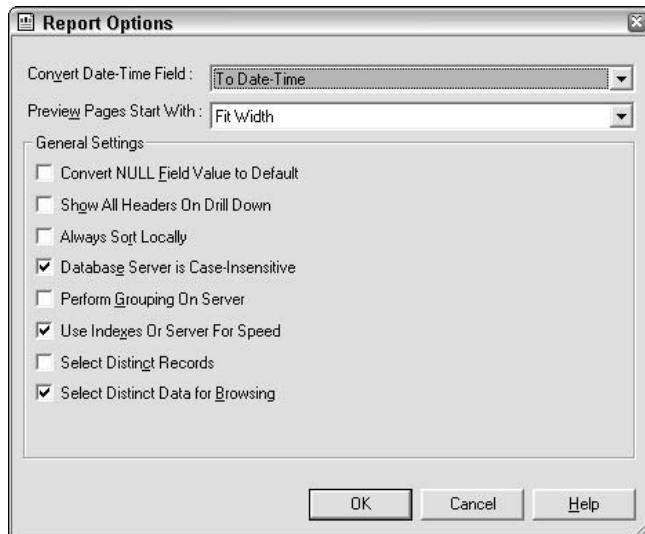


Figure 2-22

As for the page and layout attributes of your report, there are two menus that appear when you right-click your report and select the Designer option. Printer Setup and Page Setup control the orientation, paper size, margins, and so on, and they default to the settings of your default printer.

If you are working on a development machine that does not have a printer driver installed or a default printer, you may want to install one (even if you don't have a printer attached or even available) to alleviate possible problems with your report should you need to print in the future. If there is no default printer specified, there is a check box in the Printer Setup shown in Figure 2-23 (located at the top of the dialog) that is marked No Printer.

It is best practice to develop your report with multiple printers in mind, with a margin appropriate to the printer's unprintable area and no special features that would be specific to one particular printer (such as oversized paper or bleed to the edge). This will ensure that your report will print consistently on different types of printers.

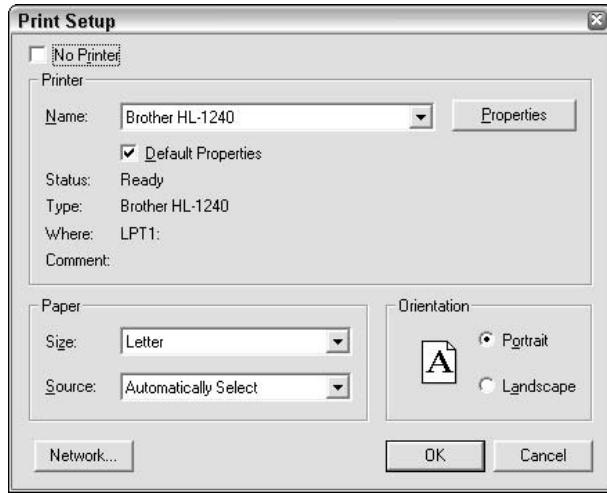


Figure 2-23

Report Design Basics

There are some basic concepts we need to get out of the way before we move on to more advanced topics. We are just going to run through the basics of how a report is put together and the options we have for controlling this framework.

Report Sections

A Crystal report comprises a number of different sections, as shown in the following screenshot (Figure 2-24). Each of these sections has a set of properties associated with it as well as a default behavior. For example, the default behavior of the Page Header section is that it will appear on the top of every page of your report.

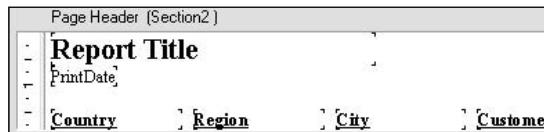


Figure 2-24

The sections of your report are clearly marked within the Report Designer by both a section name and a number. This number doesn't mean much to us now, but a little later when we want to programmatically control report sections, this notation will come in handy. To view the properties for each of these sections, right-click your report and select Format Section to open the dialog shown in Figure 2-25.

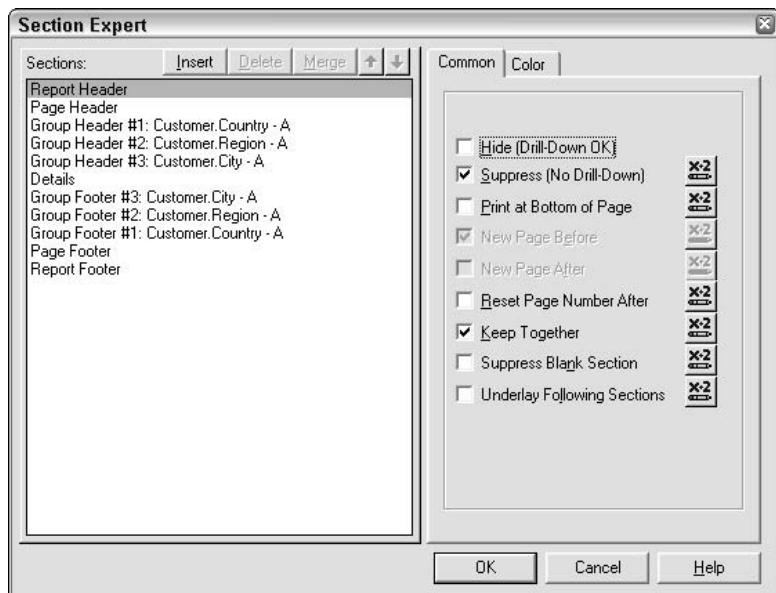


Figure 2-25

You can choose to suppress a section if you don't want it to appear, or you can change its appearance by adjusting the color or size of the section, based on your needs. Here is a rundown of the basic sections that may be contained within a report, as well as their default behavior.

Report Section	Description
Report Header	Appears on the top of the very first page of the report and is usually suppressed by default. It can be used to indicate the start of a new report or can be used as a cover sheet.
Report Footer	Appears on the bottom of the very last page of the report, is shown by default, and can be used to summarize the report (number of records, print date, and so on).
Page Header	Appears on the top of each page and can be used for column headings, the report title, and the page count, among other items.
Page Footer	Appears on the bottom of each page and can be used to display page numbers, print dates, times, and so on.
Group Header	Appears at the head of each group and is usually used to display the group name.
Group Footer	Appears at the end of a group of records and is usually used to display the group name, subtotals, or summaries.
Details Section	There's one for each record in your report. They are used to display columns of information and may be expanded to cover larger sections of fields or to create forms.

Chapter 2

You can insert multiple sections into your report in a scenario where you might want to have two different page headers, or you can separate content out for ease of use and design.

You can control all of these sections through the use of conditional formatting, which we will touch on in Chapter 8, "Formulas and Logic."

Report Formatting

Given the complex report requirements given to developers, reports can look like anything you can imagine, from a simple invoice to reports that match a preprinted form. To be honest, the majority of time spent developing a report will be in the report formatting. You probably already know where the report's data is coming from and the record selection you wish to apply; all that remains is putting it together.

There are a number of different levels where formatting can occur: You can apply some attributes by section (such as the background color or the behavior), but the majority of formatting is performed at the object level. Each object within your report will have a unique set of attributes that you can change to control the color, size, font, number, and date formats.

Although you can set these for a number of objects at once by selecting multiple objects with Ctrl-left-click or a lasso (sometimes also called a stretch-box), Crystal Reports .NET does not support the concept of grouping or classing objects together to make setting global properties easier.

If you are using your report with Web-based applications, Crystal Reports .NET does allow you to apply attributes from a style sheet to your report. For more information on how this works, check out Chapter 5, "Report Integration for Web-Based Applications."

Field Objects

Field objects within Crystal Reports .NET contain the majority of your report content. From database fields that display records, to text objects that describe each column, to the summary fields that provide the totals, any report is basically just a collection of field objects (and a little formatting).

There are eight different types of field objects that can be added to your report, and they are available from the Field Explorer (shown in Figure 2-26), which is opened by default when you open the Report Designer within .NET. (If it's not in view, remember it can be opened by pressing Ctrl-Alt-T.)



Figure 2-26

Database Fields

Database fields can be inserted from any of the tables, views, or stored procedures that appear in your report. Database fields are shown on your report using the notation of TableName.FieldName. Once a database field has been inserted into your report, a red checkmark will appear beside the field in the Field Explorer to indicate it has been used.

If you are trying to insert a field from a stored procedure and you can't see it listed in the Field Explorer, this is because by default Crystal Reports .NET will show you the tables and views only within your data source. To display stored procedures, right-click in the Report Designer. From the menu, select Designer | Default Settings and then select the Database tab. There you will find a number of check boxes for the different types of data items you can add to your report, including stored procedures.

Text Objects

Text objects are used in a report for typing text that will appear in that report, such as the column headings and comments. To insert a text object, right-click on top of your Report Designer in any section, and then select Insert → Text Object from the menu. This will insert a text object into your report in edit mode. You can type text directly into the text object. When you are finished, click anywhere outside the text object to get out of edit mode.

If you want to edit a text object already in place on your report, simply double-click the field to put it back into edit mode. If you have a large amount of text that you need to put into a text object (like an existing form letter), switch to the edit view and right-click directly on top of the text object. In the menu that appears, you will have the option to browse for and import a text file directly into the text object. You can also format a text object with tab stops, alignment options, and even line spacing by right-clicking directly on top of the object and selecting Format from the right-click menu.

Special Fields

Special fields within a Crystal Report are predefined fields that serve a specific function within your Report Designer. Examples of these special fields include page numbers, print dates, and data dates. A complete list of these special fields has been included below:

Special Field Name	Description
Print Date	The date when the report was printed
Print Time	The time when the report was printed
Modification Date	The date of the last modification to the report
Modification Time	The time of the last modification to the report
Data Date	The date when the data was read from the database
Data Time	The time when the data was read from the database

Table continued on following page

Chapter 2

Special Field Name	Description
Record Number	An internal, sequential record number assigned to all records returned to the report
Page Number	Page number
Group Number	An internal, sequential number assigned to all groups
Total Page Count	The total page count
Report Title	The report title, as saved in the report file's Summary Information
Report Comments	The comments entered in the Summary Information
Record Selection Formula	The Record Selection Formula used by the report
Group Selection Formula	The group selection formula used by the report
File Path and Name	The full path and filename of the report file
File Author	The author of the report from the report file's Summary Information
File Creation Date	The date the report file was created
Page N of M	Where N is the current page and M is the total page count

Summary Fields

Earlier in the chapter, we looked at creating a report using the Standard Expert. One of the tabs in the expert was for Total, where a summary field could be inserted into your report. Summary fields are usually associated with groups or a grand total in your report and can be inserted into your report by right-clicking anywhere within the Report Designer and selecting Insert → Summary from the menu that appears. This opens the dialog shown in Figure 2-27.

At this point, you may also notice that the right-click menu includes an option for Subtotal; subtotals and summary fields are similar, but a subtotal refers specifically to a sum, whereas a summary field could be a sum, an average, or a standard deviation.

Formula Fields

Crystal Reports .NET features a rich formula language that has evolved over the years as a powerful way to add complex calculations to reports. Formula fields appear in curly braces and are prefixed by the @ symbol; a formula used within another formula would look like this:

```
{@SalesTax} + {@InvoiceTotal}
```

Formula fields are created using the integrated Formula Editor. To see the editor, right-click Formula Fields in the Field Explorer, select New, enter a name, and then click OK to open the dialog shown in Figure 2-28.

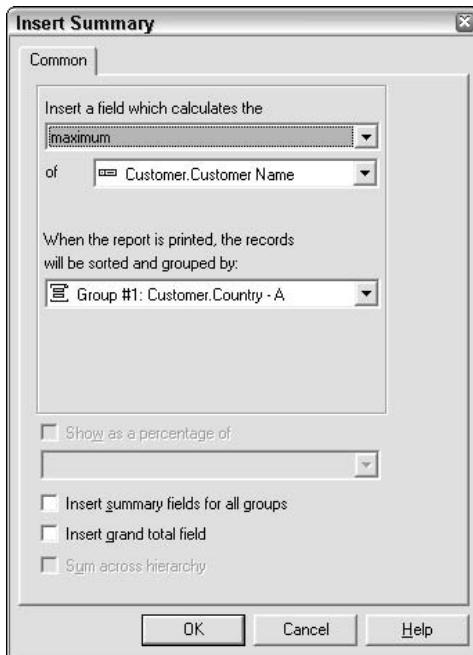


Figure 2-27

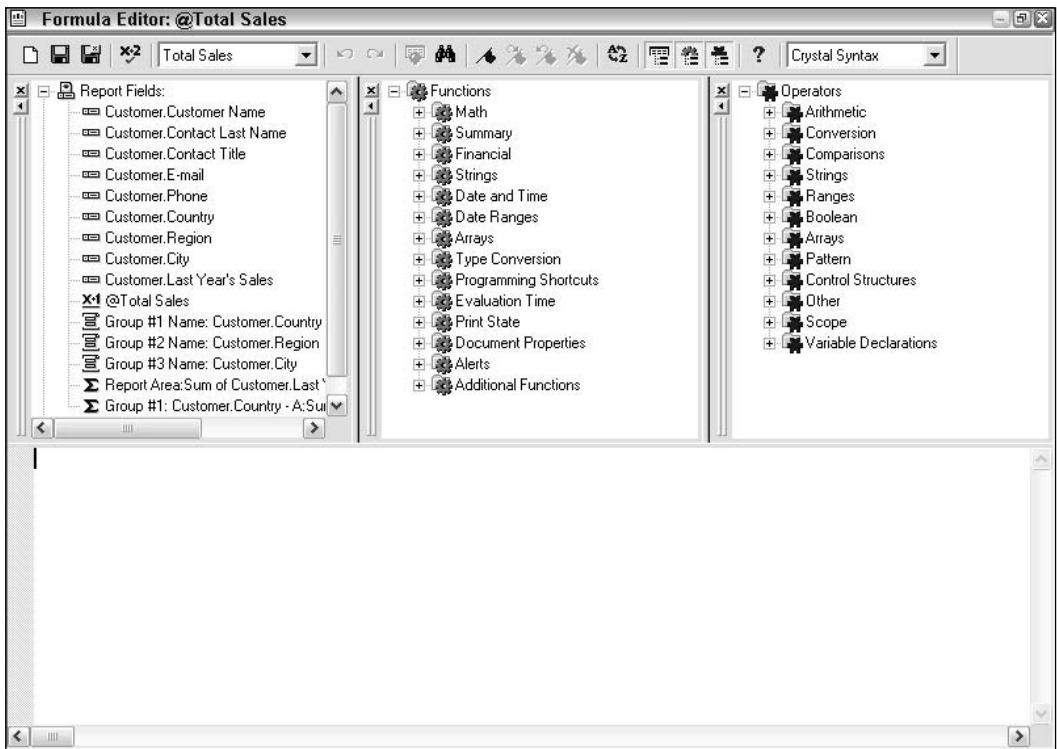


Figure 2-28

Chapter 2

When working with formula fields, you have a choice of two different types of syntax: Crystal Syntax or Basic Syntax. If you have worked with Crystal Reports before, you will probably be familiar with Crystal Syntax. It was the original formula language available with Crystal Reports and is still used for Record Selection Formulas and conditional formatting.

Basic Syntax was introduced to eliminate the need to learn a second formula syntax. The syntax, functions, and control structures are similar to Visual Basic, which many developers are familiar with, and it is easy for developers to create formulas using a language that is familiar to them.

Which language you use depends on what facet of Crystal Reports you are working with. As I mentioned earlier, the record and group selection formulas within Crystal Reports are written using Crystal Syntax exclusively, so you are going to have to learn a little bit anyway. For formulas that will appear on your report, you have a choice of using either Crystal or Basic Syntax. (You can't mix the two in one formula, but you can mix the two different types of formulas in one report.) A drop-down list in the Formula Editor controls the syntax, and you can switch between the two if required.

Formulas are covered in length in Chapter 8, "Formulas and Logic," but keep in mind you may see the Formula Editor appear in other places throughout this book; it is also used to create Record Selection Formulas and perform conditional formatting, among other things.

Parameter Fields

Parameter fields within Crystal Reports .NET are used to prompt the user to enter information when the report is run. Parameters can be used in a number of different ways, from simple data entry (like entering the name of a user to be displayed on the report) to record selection (to filter the content of a report).

Parameter fields are designated using curly braces and are prefixed by a question mark, so a parameter field in use in a formula might look something like this:

```
If {?EnterCountry} = "USA" then "North America"
```

To insert a parameter field into your report, right-click the Parameter Fields section of the Field Explorer and select New, which will open the dialog shown in Figure 2-29.

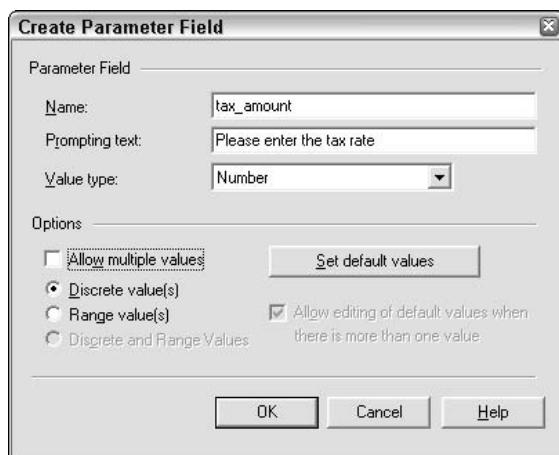


Figure 2-29

For simple parameters, you will need to give your parameter a name—the question mark prefix will be added for you—and specify some prompting text and a field type for your parameter. By default, parameter fields are set to be strings, but there are actually several different types available including:

- Boolean
- Currency
- Date
- Date Time
- Number
- String
- Time

You will also need to determine what type of values you want to be entered, such as a discrete value or a range of values, among others things.

Once you have created your formula field and inserted it into your report, Crystal Reports .NET will display a default dialog prompting the details just entered (shown in Figure 2-30) whenever your report is previewed. Most developers find this is a bit too generic for their own use and prefer to create their own interface with their own forms, including drop-down boxes, and so on. However, if you are not too concerned about how the prompt appears, this is all you need.

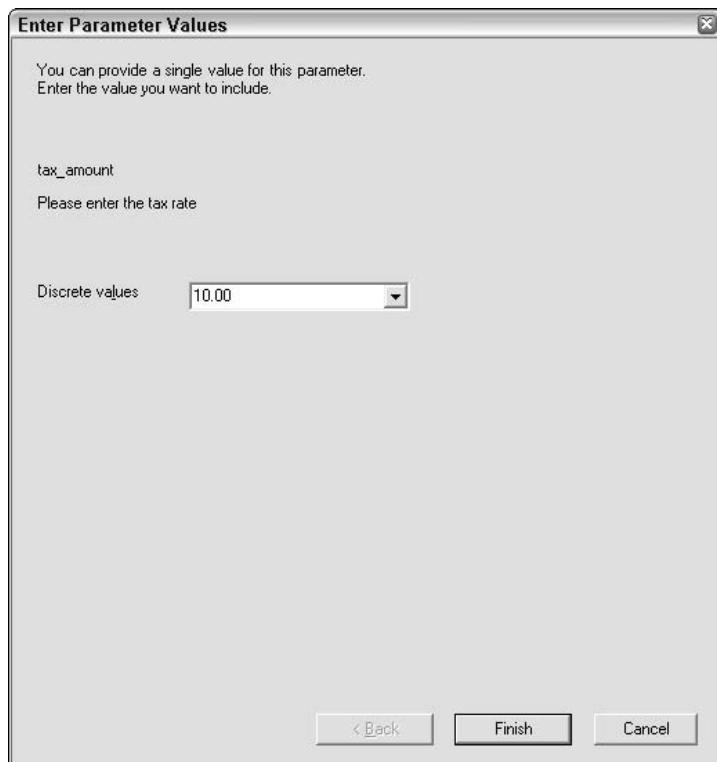


Figure 2-30

Chapter 2

The parameter field can now be used just like any other field in your report: You can place it on your report and use it in formulas. To use a parameter field with record selection, you will first need to create a parameter field to accept the input and then to set the record selection to be equal to this parameter field. For example, if you were going to prompt the user for an invoice number to reprint an invoice, you would probably want to create a parameter field called `EnterInvoiceId` and set the type to be numeric.

From that point, you would need to alter the record selection to use this parameter field. In this instance, the Record Selection Formula might look something like this:

```
{Orders.InvoiceId} = {?EnterInvoiceId}
```

When the report is refreshed, the user will be prompted to enter an invoice ID, which in turn will be passed to the SQL statement and used to retrieve the records for the report.

An import concept to remember is that the Record Selection Formula must always return a Boolean value. If the value returned is `True`, the record will be returned to the report. If the value is `False`, it will just move on to the next record.

SQL Expression Fields

In order to make the most of your database server, Crystal Reports .NET allows you to use SQL Expression instead of (or in addition to) Crystal Formulas. Using an SQL Expression field ensures that your calculation will be performed on the database server itself and gives you access to all of the database functions of SQL.

To create a SQL Expression, right-click SQL Expression Fields in Field Explorer, select New, enter a name, and select OK.

Once you have created an SQL expression field using the SQL Expression Editor, you can drag the field from the Field Explorer onto your report, and it will behave just like any other database field.

Unbound Fields

Another definite enhancement to Crystal Reports .NET is the ability to use unbound fields—that is, fields that are not tied to a specific data source. Using unbound fields, you can create a generic report and then programmatically set the content of the fields at run time. This is similar to how Crystal Reports used TTX text files in the past to hold the field structure for a data source, but Crystal Reports .NET points these fields to a dataset.

There are seven different types of unbound fields that you can add to your report:

- Boolean
- Currency
- Date
- Date Time
- Number
- String
- Time

When you drag an unbound field onto your report design, it behaves like a placeholder. When we talk about integration for Windows and Web Applications in Chapter 4, “Report Integration for Windows-Based Applications,” and Chapter 5, “Report Integration for Web-Based Applications,” you will learn how to bind data from your project with this field and display the same data in your report. For now, you need to know that until run time, these fields will look and act like formula fields (right down to an @ prefix on each), but they are actually unbound fields that will be used later.

Summary

In this chapter, we have looked at a simple report methodology and the process behind designing reports and then moved on to actually creating a report using one of the experts that ships with the product. We also had a brief look at some report design concepts and the Report Designer itself before moving to some more advanced report design topics and finally finishing up with a short overview on optimization.

This chapter covered:

- Planning your report design
- Creating a report using an expert
- Working with the report design environment
- Report design basics

With a little bit of basic report design under our belts, we are ready to move to the next chapter, where we will dig deeper into the report designer and learn some of the techniques you can use to create presentation-quality reports.

3

Designing Reports

In the last chapter we started looking at report design by creating a report, using one of the experts to walk through the report design process. Although that was probably enough information to get you started designing your own reports, the material really didn't go into some of the more advanced report design features.

In this chapter, we will be looking at the some of these advanced features, which include:

- Working with databases
- Working with groups
- Working with summaries
- Creating running tools
- Using cross-tabs
- Working with charts
- Understanding subreports
- Creating and using parameter fields

The Sample Files

In the C:\Crystal .NET2003\Chapter03\ folder, you will find the sample report and application we build in this chapter:

- CustomerOrders — a report that we will create from scratch using the Report Designer
- ViewerDemo — a small application that you can use to preview the reports in this chapter

Creating a New Report

Earlier we looked at how to create a report using the report experts. Although they provide an easy way to get started, you will sometimes want to create your own reports from scratch. To create a new report from scratch, open or create a new project, select Project → Add New Item, and select Crystal Report from the list of available templates. Enter “regionalsales.rpt” in the name field and click Open to insert the new report into your application. Just like with the report we created from the expert, a separate tab will appear for the report, and the Crystal Report Gallery will open, as shown in Figure 3-1.

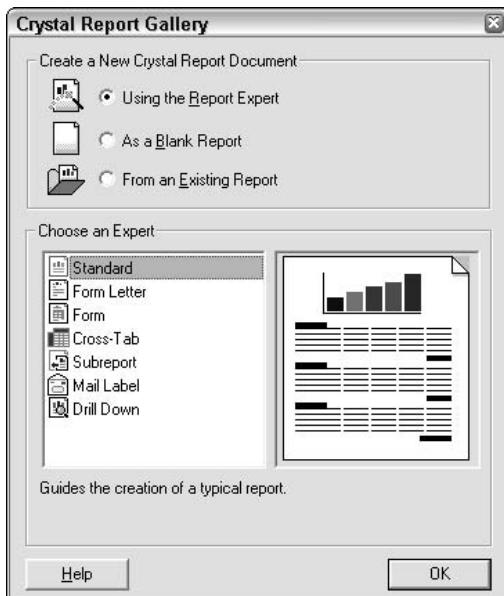


Figure 3-1

Select the option to create your report “As a Blank Report” and click OK to open the integrated Report Designer and start building your report, piece by piece.

Working with Databases and Tables

Now that you have created your report, you need to specify where the data for this report is located. To add a database or table to your report, right-click your report and select Database → Add/Remove Database to open the Database Expert shown in Figure 3-2.

There are a number of different types of data sources available, and though these were covered in Chapter 2, “Getting Started with Crystal Reports .NET,” they have been included again in the following table for your reference:

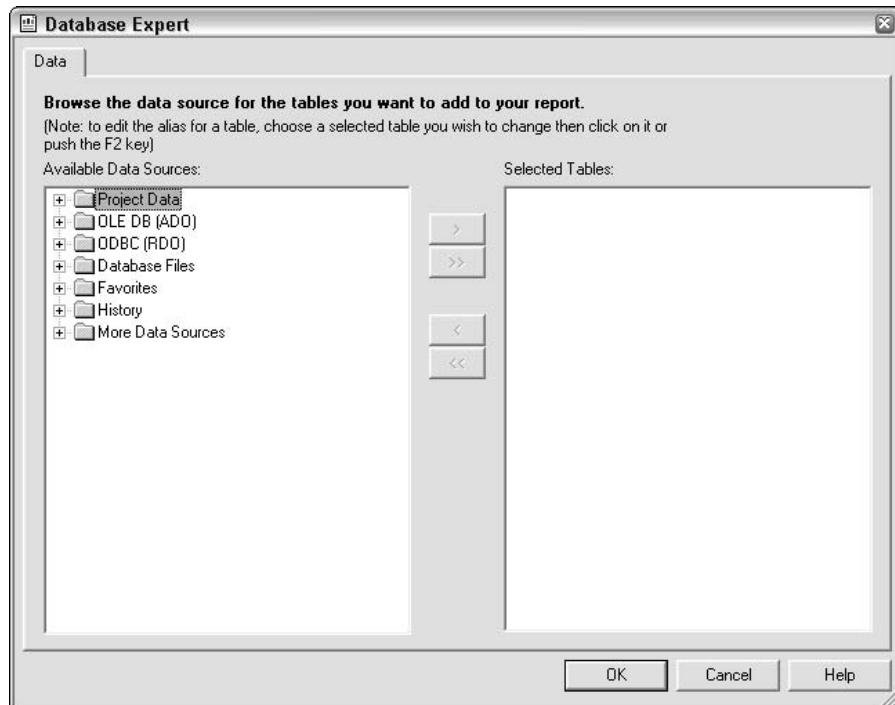


Figure 3-2

Data Source	Description
Project Data	Crystal Reports .NET can leverage the ADO .NET Framework and report directly from datasets that appear in your application. For more information on how Crystal Reports .NET can be used with ADO .NET data, please see Chapter 7, "Working with .NET Data."
OLE DB (ADO)	This folder is for data sources that can be accessed through OLE DB, including SQL Server, Oracle, and Microsoft Jet 3.51/4.00-accessible data sources (Access, Excel, Paradox, Dbase, and so on).
ODBC (RDO)	This folder is for data sources that can be accessed through an ODBC-compliant driver (which is just about every other data source). In addition to reporting from tables, views, stored procedures, and so on, Crystal Reports .NET will also allow you to enter an SQL command to serve as the basis for your report.
Database Files	This folder includes a number of file-type database formats, including Access, Excel, XML, and Crystal Field Definition files (TTX), used with previous versions of Crystal Reports and bound reporting.
More Data Sources	These include reporting directly from XML files, Access/Excel through DAO, and Crystal Field Definition Files (TTX).

Adding a Data Source to Your Report

Locate the type of datasource you would like to add to your report and expand the data source folder. Depending on what type of datasource you select, an additional dialog may open. For example, if you select ODBC (RDO), a second dialog will open and prompt you for an ODBC data source. Likewise, if you select Access/Excel (DAO) under More Data Sources, a second dialog will open and allow you to select a spreadsheet or database to report from.

In this instance, we are working with the ODBC data source labeled “Xtreme Sample Database 2003,” so once you have selected this data source, your Database Expert should look something like Figure 3-3.

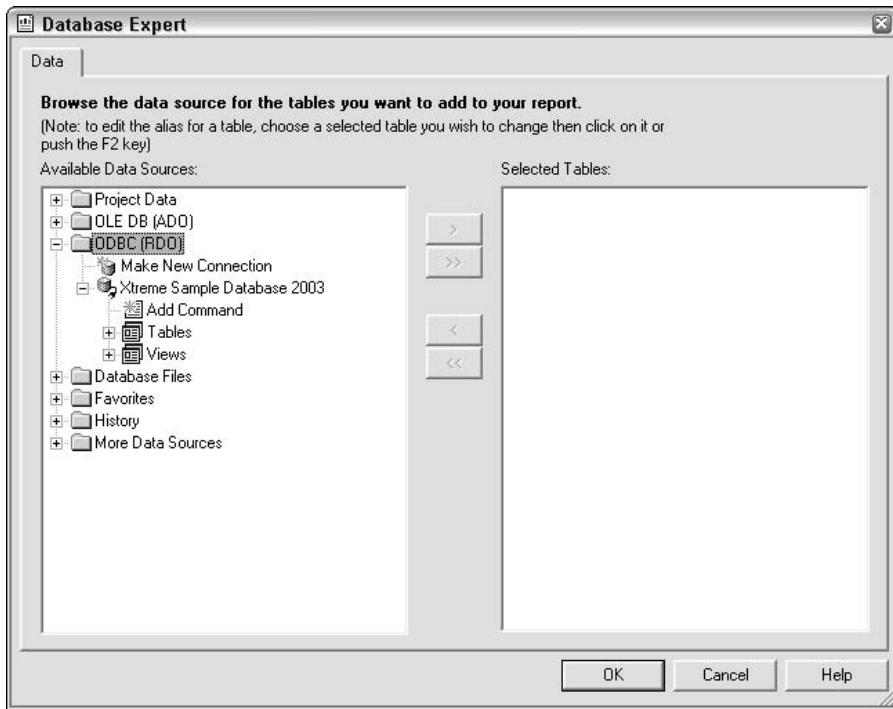


Figure 3-3

There are three additional nodes shown underneath your datasource: The Tables and Views nodes contain a list of all of the tables and views in the data source you have selected, and the Add Command option allows you to enter a SQL statement to serve as the basis for your report, which we will look at a little later.

You can also report off of stored procedures, system tables, and other objects. To turn this option on, right-click your report and select Designer → Default Settings. Under the Database tab, click the show options for the types of objects (tables, views, stored procedures, and so on) you want to report from.

In this case we will be reporting off of tables within our sample data source, so expand the Tables node, select the Customer table, and then use the right-arrow icon to add it to the list of selected tables. Likewise, select the Orders table using the same process.

Because we have selected two tables, an additional Links tab will appear on the Database Expert (shown in Figure 3-4). This tab will allow us to specify the relationship between these two tables.

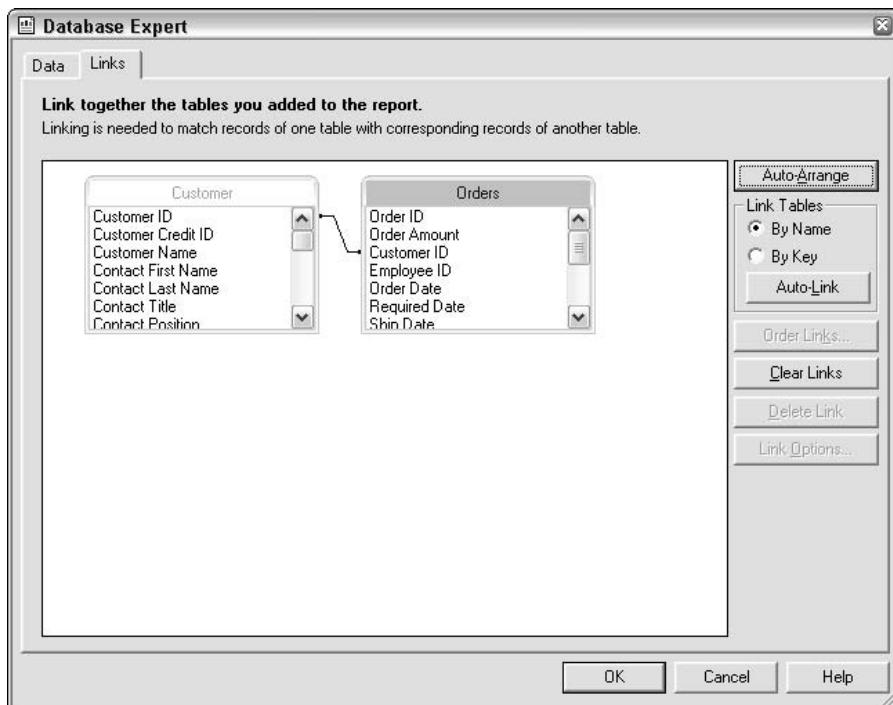


Figure 3-4

Using the options shown in the Links tab, you can draw links or joins between the databases and tables in your report to indicate the relationship between each. To specify a link between two fields, drag the first field and drop it on top of the second.

If you make a mistake, you can remove a link by clicking the line to highlight it and then pressing the Delete key or, to clear all links, clicking the Delete button on the right side of the expert. This option is especially handy when Crystal Reports automatically attempts to use smart links in the tables you have selected.

By default, Crystal Reports will join two SQL tables with an Inner Join. To change the default join type, right-click directly on top of the line drawn between the two tables and select Link Options from the right-click menu to open the dialog shown in Figure 3-5.

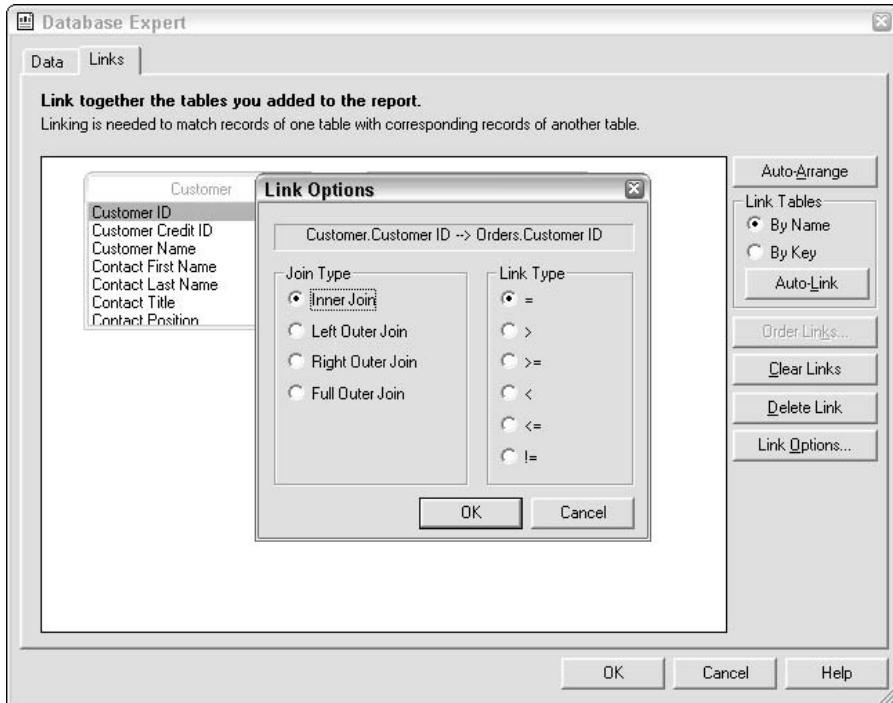


Figure 3-5

Using the Link Options dialog, select a join type for this link from the list. Your choices are:

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

You can also use this dialog to select a link type using one of the comparison operators (=, >, >=, <, <=, and !=). When you are finished specifying the join type, click the OK button to return to the Links tab.

If you are working with a large number of tables, you may want to consider using the Auto-Arrange option found on the right-hand side of the Links tab. Click the Auto-Arrange button, and Crystal Reports will arrange the tables within this dialog to make viewing the layout and joins a bit easier.

By default Crystal Reports will auto-link the tables you have selected for your report, based on the field name and key fields found in the tables. To use this option yourself, select either the By Name or By Key options within the Links tab, and then click the Auto-Link button.

If you are unsure about how accurately this will link the tables in your report, don't forget that you can always remove individual links by highlighting the link to be deleted and then either pressing the Delete key on your keyboard or clicking the Delete Link button. You also have the option of removing all links

by clicking the Clear Links button. If you have multiple links, you can use the Order Links button to specify in what order these links will be evaluated.

With your database linking in place, click OK to return to your report design. You can now start adding fields to your report. At the end of the last chapter, we had a look at the different types of fields you could add to a report. For the sample report we are working on, we want to add some database fields, as we will be using these fields later to demonstrate some advanced reporting features.

To do so, open the Field Explorer and drag the following fields onto your report's detail section:

- Customer.Customer Name
- Customer.City
- Customer.Region
- Customer.Country
- Orders.Order Date
- Orders.Order Amount

Your report should now look something like Figure 3-6.

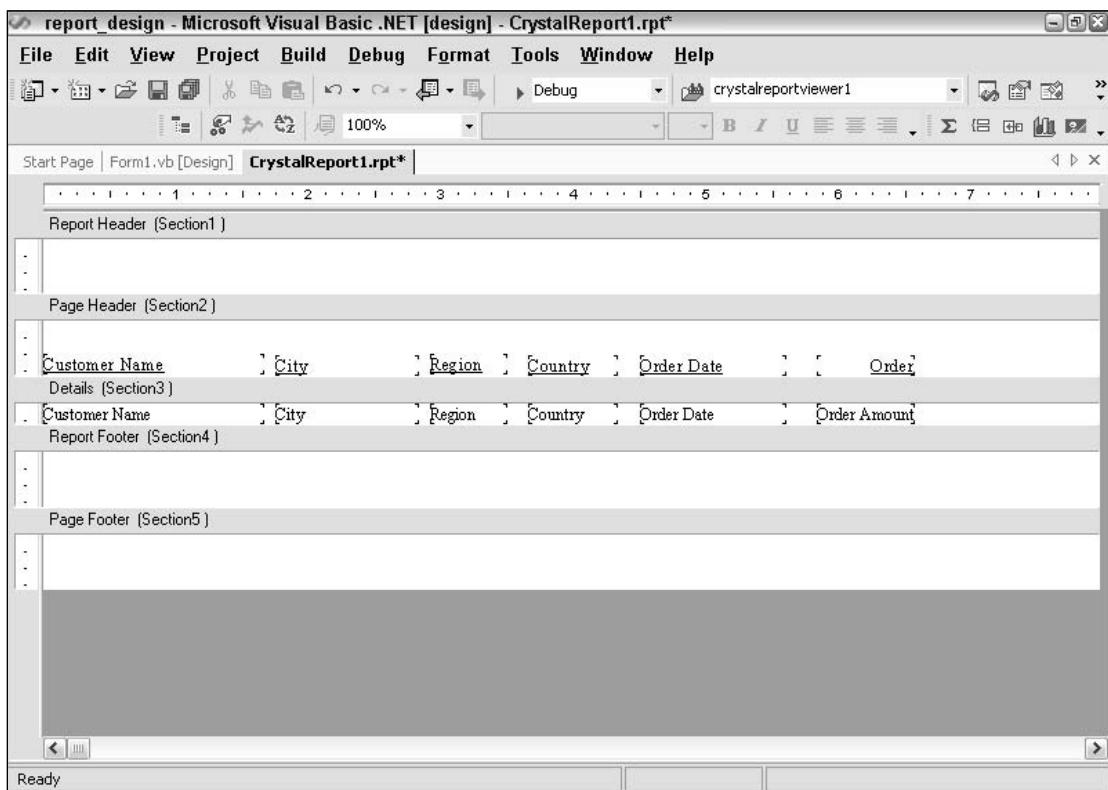


Figure 3-6

If you ever need to remove a table or database from your report, it is pretty simple: Select Database → Add/Remove Database, highlight the table you want to delete, and use the left arrow to remove it from the list of selected tables. If fields from the database you are trying to remove have been used in your report, you will receive the warning message "There are fields in the report from this file. Continue?" Click OK to continue and remove the file, or click Cancel to leave the database or table in your report. When you are finished, you will be returned to your report design.

Setting the Data Source Location

Another handy feature is the ability to change the database location of your reports. You can design a report on your test database, for example, and then later point it to a production database where your data resides. To change the database location of your report, select Database → Set Location to open the dialog shown in Figure 3-7.

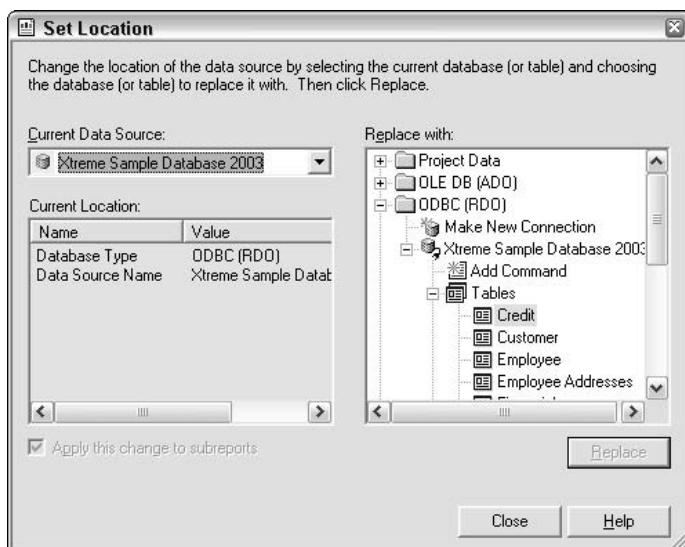


Figure 3-7

Use the drop-down list to select the Current Data Source you would like to change, and then locate the replacement data source in the list of data sources on the right. Click to highlight the replacement data source, and then click the Replace button to make it happen.

If you have subreports that use the same data source and you want to switch them over at the same time, leave the check box in the bottom left corner of the dialog checked.

If you are using multiple databases or tables in your report, a prompt will appear with the question "Propagate database and server changes across tables with the same original information?" Clicking Yes sets the location for all of the other databases or tables in your report; clicking No changes the location of only the one particular database or table you have chosen.

When you are finished setting the location, click the Close button to return to your report design.

If the data structures differ between the old database or table and the new location you have selected, a Map Fields dialog will appear, and you will have to map any unfound fields in your report to fields in the new database structure.

Verifying Database Structures

As your database structures evolve and change, reports you have created from these structures may no longer work because of different field names, types, and so on. To ensure that the changes made in the database are reflected and accounted for in your existing reports, you will need to verify the database that they were created from after any changes by selecting Database → Verify Database. If you have databases or tables in your report that are not used, you may receive the message “Verify files in report that are not used?” Click Yes to proceed.

At this point, Crystal Reports will run through the data structures in your report and verify that nothing has changed. If all of the data structures are unchanged, you will receive the message “The database is up to date.”

If anything has changed in the data structures, you will receive the message “The database file xxx has changed. Proceeding to fix up the report.”

If Crystal Reports finds simple changes, such as a database field that has been extended or a decimal place that has changed, it will simply update its version of the data structures and display the message “The database is up to date.”

If Crystal Reports finds a major change (such as a missing field name or a changed field type), it will open a field mapping dialog and allow you to map fields from the old data source that are not found in the new data source. (This often happens when field names change.)

With a little database work out of the way, we can move on to adding some extra value to your reports through grouping, which is what we will look at in the next section.

Working with Groups

We looked at groups briefly in Chapter 2, “Getting Started with Crystal Reports .NET,” as this was an option in the expert that we used to create your first report. In Crystal Reports, *grouping* is used to place similar records together. For example you may have a Products table in your database that lists all of the products your company sells, including information like the product name, product type, product class, and so on. For example, if you were to insert a group based on the product class field, all of the records that had the same product class would be grouped together, like in the report shown in Figure 3-8.

Within a report, groups are used to organize your report data to make the report more readable and to make it easier to find information quickly. In addition, groups also allow you to create summaries that will give you the values such as the sum, average, maximum, and minimum within that group. In the following sections we are going to look at how to insert and format groups, and then a little later we will look at adding summaries and calculations based on these groups.

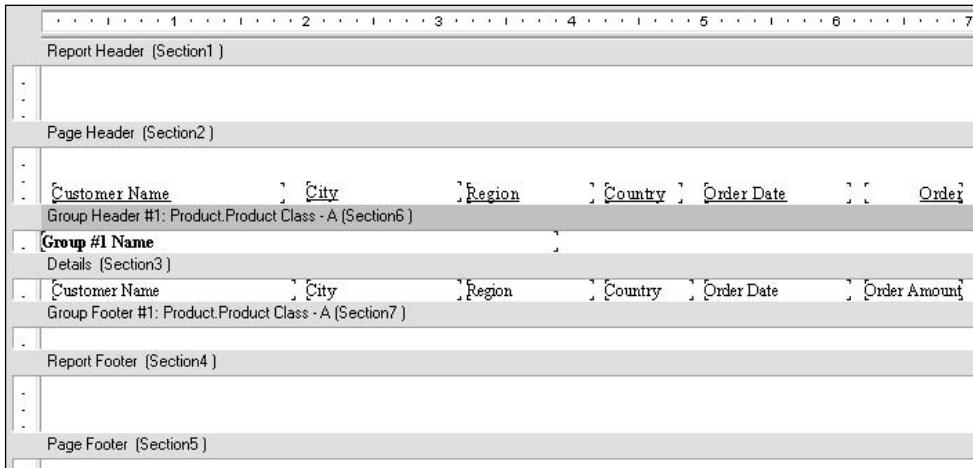


Figure 3-8

Inserting a New Group

To insert a new group into your report, right-click any blank area within your report and select Insert → Group. This opens the dialog shown in Figure 3-9.

The first choice you will need to make is which field you want to group on. This can be a database field, a formula field, or an SQL expression. Use the drop-down list provided to select a field. The fields that have been inserted onto your report will appear at the top of the list, and the fields from the database tables that have been added to your report will follow, regardless of whether these fields are used in the report or not. Remember, you don't need to actually show a field on your report to use it in a group.

Once you have selected the field you want to use, you will need to select the order for your group; by default, this is set to Ascending, which means your groups will be arranged from A–Z, 1–9, and so on. In Chapter 2, “Getting Started with Crystal Reports .NET,” we had a look at the different sorting orders available for groups, but here they are again for reference.

Sort Order	Description
Ascending	For ordering the data from A–Z, 1–9, and so on.
Descending	For ordering the data from Z–A, 9–1, and so on.
Original	If your dataset is already sorted, this option will leave the data in its original sort order.
Specified	Used for creating your own custom groups and setting some criteria. Any records that meet the criteria would belong to the specified group.

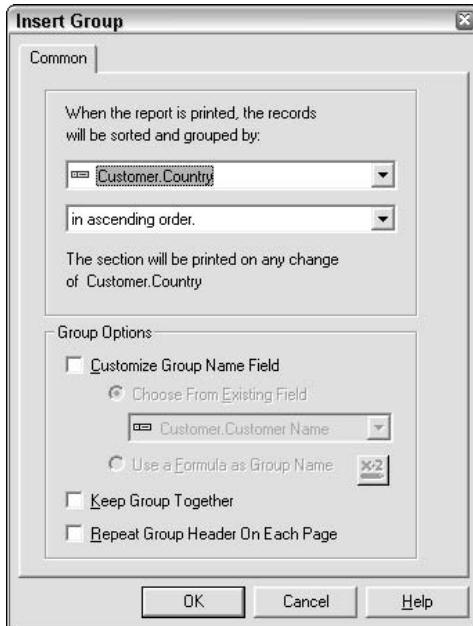


Figure 3-9

And finally, at the bottom of the dialog are some grouping options that are optional. We'll look at these a little later in this chapter.

If you had selected a date or date-time field for your group, you would also have an additional option to set for how the groups would be broken down (by day, week, month, quarter, and so on).

For now, all you need to do to create a group is to select a field, pick a sort order, and click the OK button. If you were to add a group to the report we have been working on by using the Customer.Customer Name field, your group would be inserted onto your report, which would look something like Figure 3-10.

You'll notice that there are two additional sections added to your report for the group header and group footer, and a special field labeled "Group #1 Name" has been inserted in the group header.

This special field is called a *Group Name* field and is used to label the groups within your report. Remember in our earlier example when we had a group on a Product Class field? The Group Name field is used to display the name of each group, so from our earlier example it would display "Accessory" for the first group and then "Bicycle" for the second. Although this is a special field, you can customize the name that appears for each group, which we will also look at a little later in this chapter.

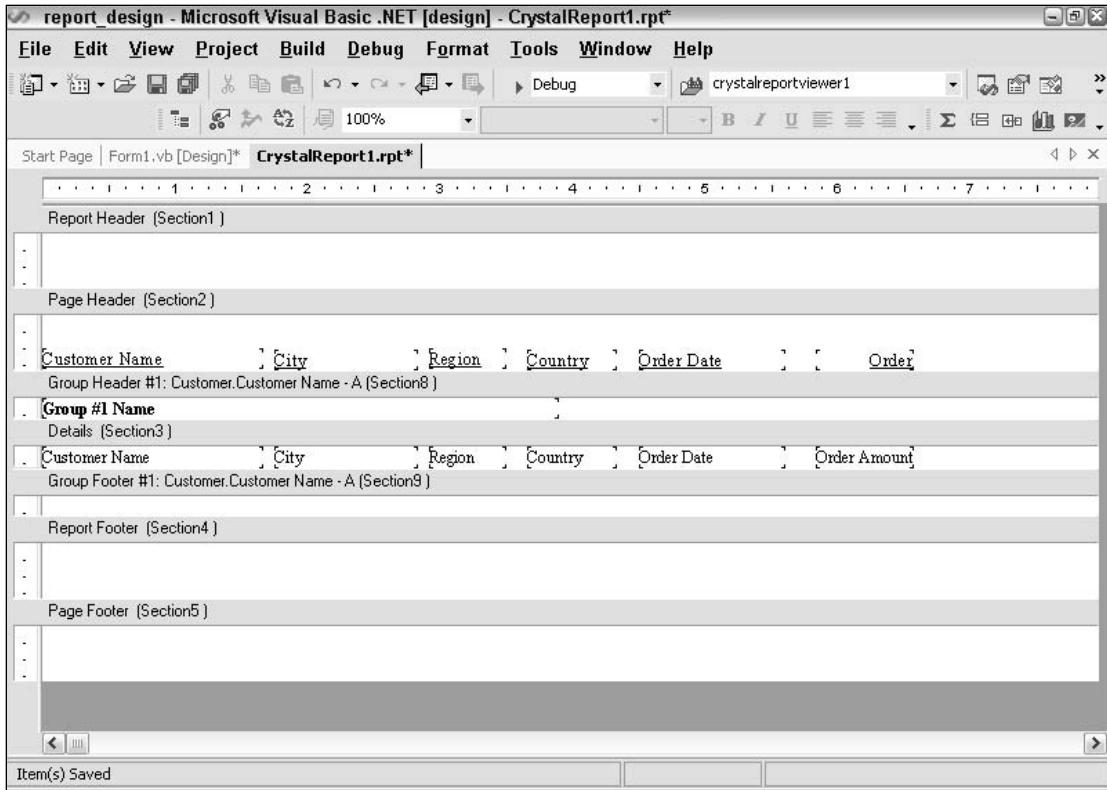


Figure 3-10

Changing Groups

To change the existing groups that appear on your report, right-click the group header or footer and select Change Group from the right-click menu to open the Change Group Expert shown in Figure 3-11.

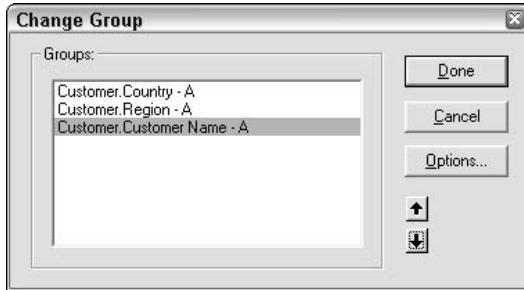


Figure 3-11

Select the group you want to change from the list and click Done, which opens a dialog similar to the one used to create the group. You can also change the order that groups appear in your report by using the up and down arrows on the right-hand side.

Deleting Groups

To delete a group from your report, locate the group header or footer, right-click directly on top of it and select Delete Group. This will delete the group, including the group header and footer and, more importantly, all of the fields that are present in either of these sections. So if you want to preserve the formatting you may have applied to this section or the fields it contains, move these fields out of the section before deleting. If you do delete the group by mistake, you can use Undo (Ctrl-Z) to undo the delete and get the group back.

Formatting Groups

Once you have groups inserted into your report, the easiest way to format them is through the Group Name field. This special field is placed on your report automatically whenever you create a group and can be formatted just like any other field: You can change the font, size, and so on. If you have deleted your Group Name field or want to put it in a different location, you can insert a new Group Name field by locating the field in the Field Explorer under the Special Fields section, as shown in Figure 3-12.

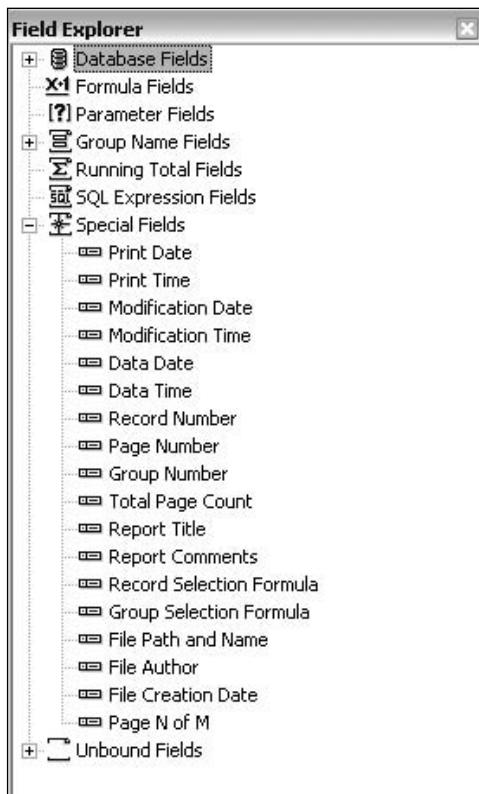


Figure 3-12

Chapter 3

You can also customize the group name field by right-clicking in the group header or footer and selecting Change Group from the right-click menu. Then select the group you want to work with and click Options to open the dialog shown in Figure 3-13.

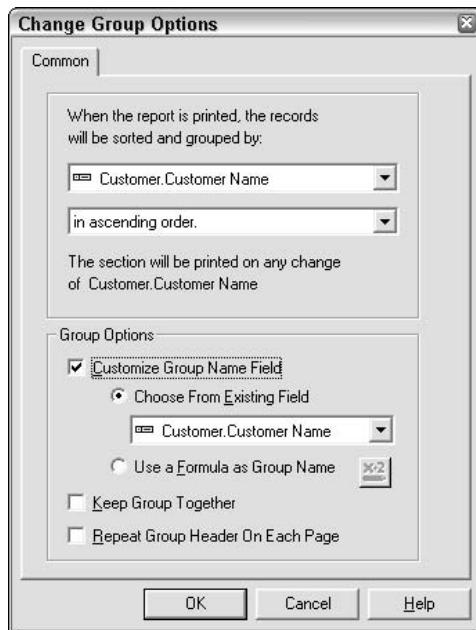


Figure 3-13

By default, Crystal Reports will automatically create the group name for you based on the field you are grouping on, but you can override this option. In the bottom of the dialog, select the Customize Group Name Field check box. For selecting a new group name, you have two options. If the group name that you want to use is stored in a database table and has a direct relationship to your group field, you can use the drop-down list to select an existing field.

An example of this would be a report that grouped on Product Type and was a reference number (for example, 001, 002, or 003). If you had a lookup table that stored the reference number and the description of the product type (that is, 001 = "Mountain Bikes", 002 = "Racing Bikes"), you could use this table to display a proper name instead of the product type number.

If you don't have a table that stores this information or if you want to do a more complex lookup for the group name, you can select the option immediately below to "Use Formula as Group Name" and then click the X+2 icon to open the formula editor and enter a formula to derive the group name.

For more information on working with formulas, check out Chapter 8, "Formulas and Logic."

This formula should take the format of an if..then statement with the output being the group name itself (for example, if {Customer.Country} = "USA" then "United States").

You may also notice there are two additional settings at the bottom of the Change Group Options dialog. The Keep Group Together setting is designed to help eliminate orphaning of groups across multiple pages. (This sometimes works with mixed results.) The second option, Repeat Group Header On Each Page, will repeat the header section of your group on every page of your report. This can be handy for large reports that have multiple pages of the same group; with the group header at the top, you will always know where you are in the report's structure.

And finally, the last formatting option we probably need to look at is how to throw page breaks between groups. You won't find this option on any of the dialogs we have used to create or edit groups. This setting is related to the section formatting options we looked at in Chapter 2 when we had our first look the Section Expert. Right-click either the group header or footer and select Section Expert from the right-click menu to open the dialog shown in Figure 3-14.

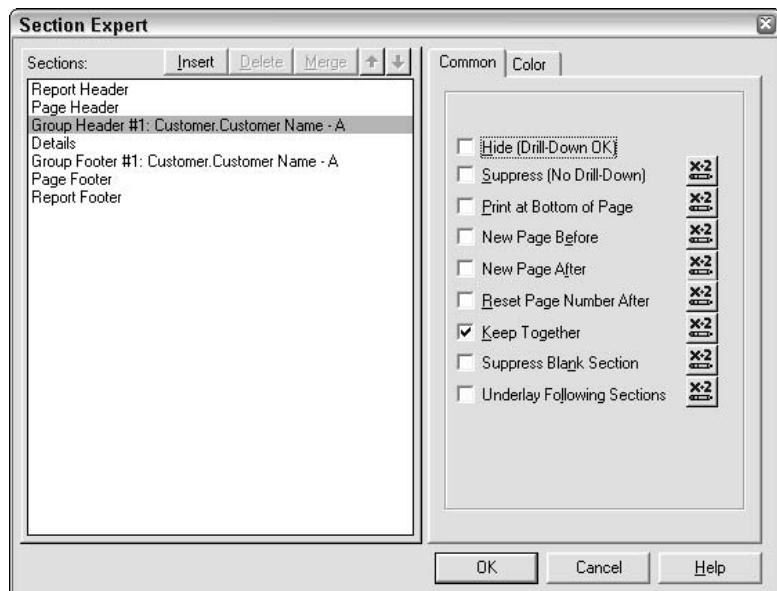


Figure 3-14

Select the section where you want to throw the page break and then use the check boxes on the right side of the dialog to select either the New page before or New page after option, depending on your requirements.

Record Sorting

Another tool that you have to help organize your reports is *record-level sorting*. Record-level sorting is similar to grouping in that it orders the records according to the criteria you specify (for example, alphabetically by country). However, the difference is that record-level sorting leaves the records in one big group. In the earlier example of grouping, the records were grouped by country and put in order so that each country had its own group, group header and footer, and so on. If we had performed a sort on those same records, they would have been ordered but left in one large list.

Chapter 3

To add record-level sorting to your report, right-click your report in a blank area and select Report → Sort Records to open the dialog shown in Figure 3-15.



Figure 3-15

Select the field(s) that you want to use for your sort and move it from the list on the left to the list on the right by highlighting the field and clicking Add. In the report we have been working through in this chapter, add the field Orders.Order Date so that all of the records within the group will be ordered by the date the orders were placed.

If you want to change the sort order, you will need to remove all of the fields and then add them back in the correct order.

Working with Summaries

Simple summary fields include sum, average, minimum, maximum, and other calculations that do not require any additional criteria. Summaries are most often used with groups and will appear on the group footer if a group has been selected. If you selected the option to create a Grand Total, these will always appear in the report footer on the very last page of your report.

Inserting a Summary Field

To create a simple summary field, click the field that you want to summarize and select Insert → Summary to open the dialog shown in Figure 3-16.

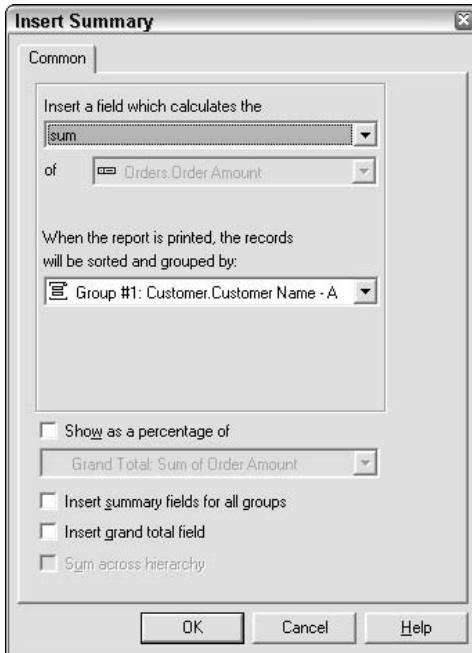


Figure 3-16

You will need to select a summary operator from the drop-down list at the top of the dialog, as well as which group the summary should be inserted into. If you would like this summary to be calculated as a percentage of a grand total, you can use the check box at the bottom of the dialog to enable this option and the drop-down list to select the grand total field. This feature is especially handy when used alongside sums and averages. For instance, if you were creating a report on international sales, you could show the dollar amount (sum) for each country and the average sales, as well as a percentage value representing that country in the total sales.

Another real time-saver is the option for inserting summary fields for all of the groups in your report; this will copy the summary field to the group footers for all of the groups inserted onto your report. And finally, the last option is for inserting a grand total field, which would appear at the bottom of the report in the report footer. (You can also insert grand totals independently from the right-click menu you used to insert the summary field.)

Using the report we have been creating in this chapter, add a summary field to the Orders.Order Amount field to calculate the sum. Your report should now look something like Figure 3-17.

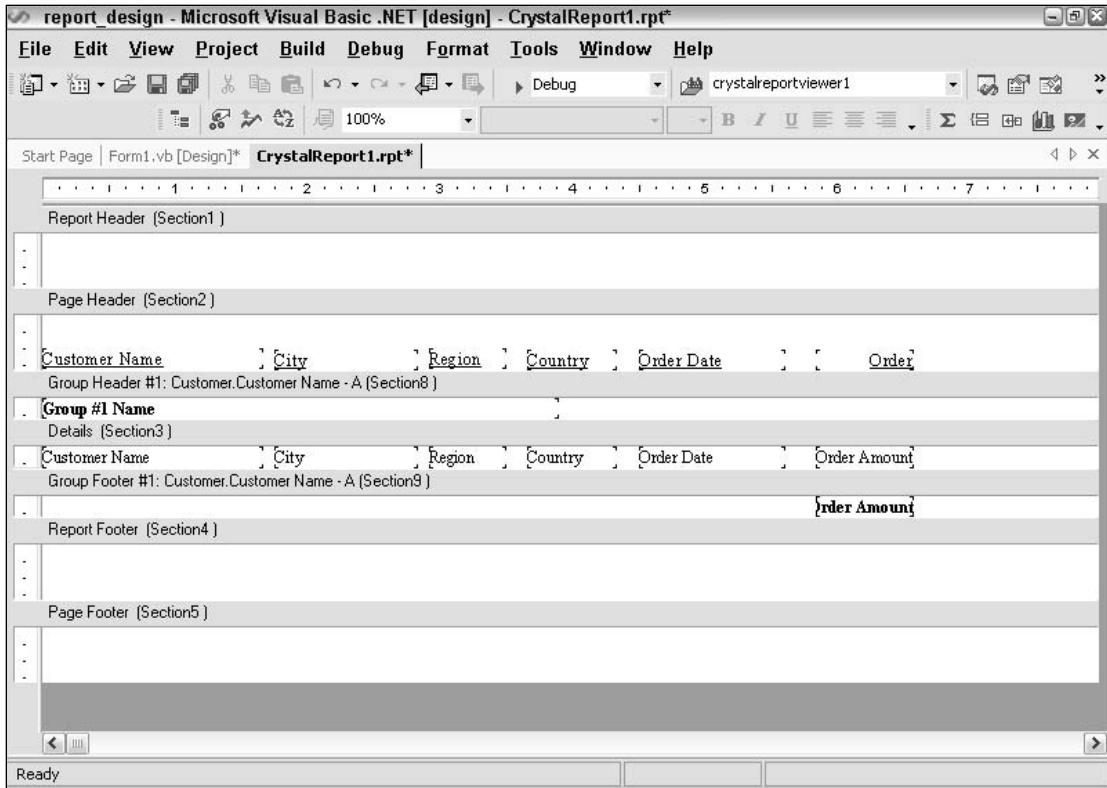


Figure 3-17

Changing a Summary Field

After a summary field has been inserted into your report, you can change the summary operation by locating the summary field that you want to change on your report and selecting Change Summary Operation from the right-click menu. In the Change Summary dialog, use the drop-down list provided to change the summary operation and then click OK to accept your changes. The operator change should immediately be reflected in your report.

TopN/Group Sorting

TopN/Group Sorting is a function of Crystal Reports used to sort groups according to a summary field that has been created based on that group. This function is often used to determine something like the top 20 customers (that is, Top N, where N is 20) or the top or bottom five products, and it can also be used to order groups based on a summary field. Before you can use TopN/Group Sorting in your report, you need to make sure that you have two things inserted onto your report: a group and a summary field. Without both of these present, you cannot use TopN/Group Sorting.

To add TopN/Group Sorting to your report, verify that your report has at least one group and one summary inserted, and then right-click a blank area of your report and select Report → TopN/Sort Group Expert to open the dialog shown in Figure 3-18.



Figure 3-18

Using the drop-down list at the top of the dialog, select the type of analysis you want to perform from the following list:

- All (for placing the groups in order by a particular summary field value)
- Top N
- Bottom N
- Top Percentage
- Bottom Percentage

By default, the other groups in your report are included in a group called Others. If a group is not included in the Top (or Bottom) N or the percentage you specify, it will get lumped into this group. To suppress this function, click the check box at the bottom of the dialog. If you would like to keep the Others group but just change the name of the group, you can enter a new name in the text box beside the Include Others box. Click OK to accept your changes and to apply these changes to your report.

In the report we have been working through in this chapter, apply TopN analysis to get the Top 10 customers based on the Order Amount field and discard the Others. Your report should now only show the first 10 customers, as shown in Figure 3-19.

You can also use this type of analysis to sort the groups in your report according to some summary field that you have inserted in your report. This functionality is similar to TopN analysis, except that it isn't limited to just a certain number of groups. To sort the groups in your report by the value of a summary field, repeat the same process we just went through, but instead of selecting TopN, choose the option for All, and your groups will be ordered by the summary field you select.

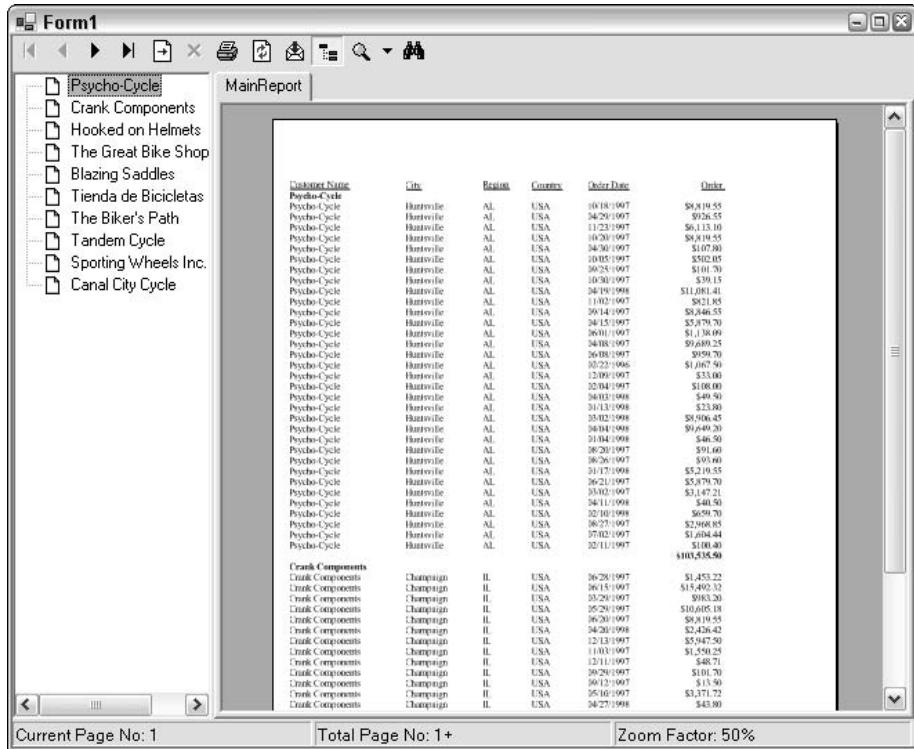


Figure 3-19

Using Running Totals

Running totals provide an at-a-glance look at cumulative values in your report and display a running summary beside each record. With each release of Crystal Reports, running totals have grown in functionality, and they can now be used for a wide range of summary and analysis tasks. Running totals also feature a flexible evaluation and reset function that makes complex analysis easier.

By using running totals in your report, you can quickly give users the information they need without having to wade through the report to get to a summary field or the end of a section. For example, suppose you want to create a running total that runs along side a list of your customers and their last year's sales. With each record, you want this last year's sales figure added to the running total, as shown in Figure 3-20.

The first step in creating a running total is to locate and select the field that will serve as the base field. Right-click directly on top of the field and select Insert → Running Total, which opens the dialog shown in Figure 3-21.

The screenshot shows a Windows application window titled "Form1". On the left is a navigation pane with a tree view containing nodes like "Psycho-Cycle", "Crank Components", "Hooked on Helmets", "The Great Bike Shop", "Blazing Saddles", "Tienda de Bicicletas", "The Biker's Path", "Tandem Cycle", "Sporting Wheels Inc.", and "Canal City Cycle". The main area displays a table with three columns: "Order Date", "Running Total", and "Order Amount". The table contains 15 rows of data. At the bottom of the window, there are status bars for "Current Page No: 1", "Total Page No: 1+", and "Zoom Factor: 150%".

Order Date	Running Total	Order Amount
0/18/1997	\$8,819.55	\$8,819.55
0/29/1997	\$9,746.10	\$926.55
1/23/1997	\$15,859.20	\$6,113.10
0/20/1997	\$24,678.75	\$8,819.55
0/30/1997	\$24,786.55	\$107.80
0/05/1997	\$25,288.60	\$502.05
0/25/1997	\$25,390.30	\$101.70
0/30/1997	\$25,429.45	\$39.15
0/19/1998	\$36,510.86	\$11,081.41
1/02/1997	\$37,332.71	\$821.85
0/14/1997	\$46,179.26	\$8,846.55
0/15/1997	\$52,058.96	\$5,879.70
0/01/1997	\$53,197.05	\$1,138.09
0/08/1997	\$62,886.30	\$9,689.25

Figure 3-20

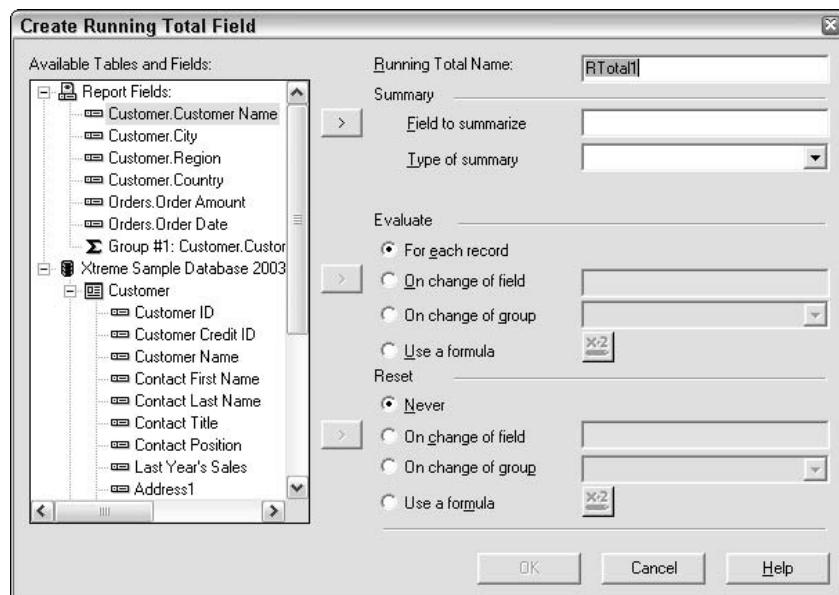


Figure 3-21

Chapter 3

The first thing you need to do is type a name for your running total in the Running Total Name box. It can be any name you like, as long as it makes sense to you. Crystal Reports will put the pound symbol (#) in front of your running total name so you can easily identify this field as a running total when it is inserted into your report.

The next step in creating a running total is to select a field to summarize and then choose a summary option. To select a field, locate it in the list on the left and then click the top right arrow to move the field to the textbox on the right. From the pull-down box immediately below the summary field, you need to select a summary operator. All of your old favorites are here: Sum, Average, and so on. And because in this example you are inserting a running total that will run down the page, you don't need to worry about the Evaluate and Reset options for the running total. Click OK to accept your changes and return to your report design. You'll notice that your running total field has been inserted into your report in the Details section.

As you created your first running total field, you may have noticed two sections in the Create Running Total Field dialog marked Evaluate and Reset. These sections are for setting the options related to when your running total will be evaluated and when the total will be reset. For evaluation times, you can select a calculation time for your running total from the following options:

- For Each Record
- On Change of Field
- On Change of Group
- Use a Formula

For example, you would want to use these options if you were creating a running total to sum all of the international sales in a report. You could select the Use a Formula option and enter this criteria:

```
{Customer.Country} <> "USA"
```

The resulting running total field would be evaluated only for those customers who are not in the United States.

Likewise, you can reset your running total field using the following options:

- Never
- On Change of Field
- On Change of Group
- Use a Formula

For example, you could reset the running total for each change of the country field within your list. By using the evaluation and reset options, you can create running total fields for just about any use you can imagine.

To use these options, in most cases you will need to select the option and the corresponding field or group. For the Use a Formula option, you will need to select the option and then click the X+2 button to open the Crystal Reports formula editor and enter your criteria. As with record selection, the formula

you create here needs to return a Boolean value — either true or false. If the value is true, the record will be evaluated or the running total reset (depending on the option you are working with); likewise, if the condition evaluates to false, the action will not take place.

For more information on working with formulas, check out Chapter 8, “Formulas and Logic.”

Using Cross-Tabs

Cross-tabs within Crystal Reports .NET can be used to display summarized data in rows and columns, similar to a spreadsheet. If you want to create a report with a cross-tab as its main feature, there is a Cross-Tab Expert available that will guide you through the steps to do so.

You can also insert a cross-tab into an existing report, which we will walk through here. You may want to create a copy of the report we have been working with before starting this section. That way you can always return to the original, which we will use a little later for charting. To insert a cross-tab into your report, right-click the report and then select Insert → Cross-Tab from the menu, as shown in Figure 3-22.

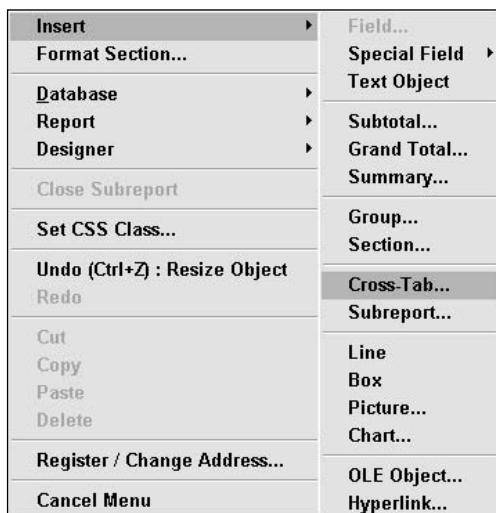


Figure 3-22

This makes a box appear over your cursor, which you should position over your report where you want it to appear. Place the cursor over the Group Header #1: Customer.Country – A section, next to the Group #1 Name label. Left-click the mouse, and the dialog shown in Figure 3-23 will open. When working with a cross-tab, there are three basic elements: rows, columns, and summarized fields. To start, you will need to select at least one of each using this dialog.

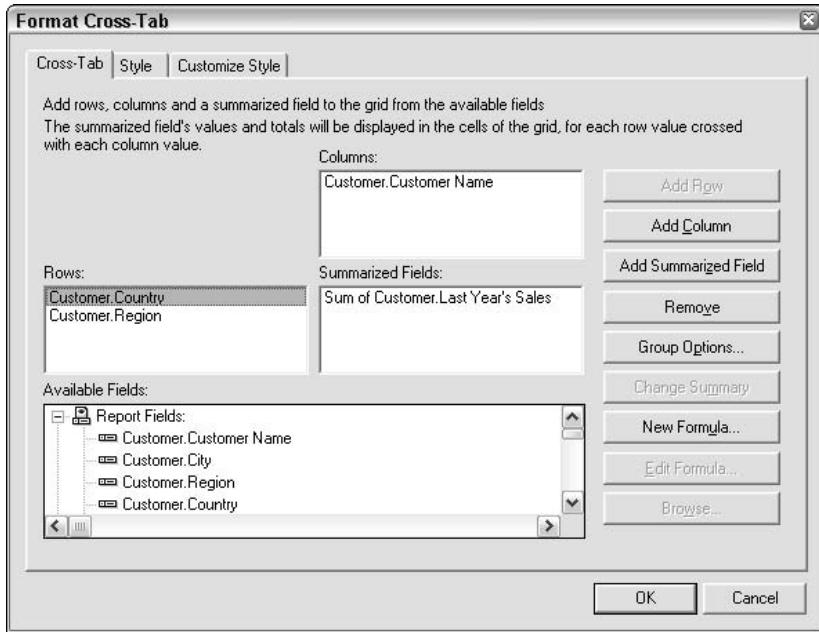


Figure 3-23

Select Customer.Country for the row and Customer.Region for the column. Select Customer.Last Year's Sales so it is highlighted, and then click the Add Summarized Field button so Sum of Customer.Last Year's Sales appears in the summarized field. Click OK to create this cross-tab.

In the first section under each country, your report now has a summary of the last year's sales by region. In the screenshot shown in Figure 3-24, we have the sales by region for the USA. If you select England in the list to the left of the report, the same cross-tab will appear with values for the regions in England.

By default, Crystal Reports .NET applies some standard formatting to your cross-tab and draws lines and boxes around the columns and rows. With the use of a style sheet, you can apply a predefined style to your cross-tab, as shown in Figure 3-25, or simply change individual attributes to suit.

Cross-tabs have come a long way since the early version of Crystal Reports. Now if the data runs off the page, it is continued on the next page and a margin is used so the data is not cut off, eliminating the old problem of taping multiple report pages back together.

The only down side is that there are still some issues with page numbering. If your cross-tab runs across multiple pages, the page number will not work correctly (for instance, if it is three pages across and three down when printed, only the first and every third page will have a number, and it will be 1...2...3). Still, those limitations are minor in comparison to the functionality provided.

Form1

The screenshot shows a Windows application window titled "Form1". On the left, there is a tree view control displaying a hierarchical list of countries. The root node is collapsed, and its children include Argentina, Aruba, Australia, Austria, Bahamas, Bangladesh, Barbados, Belgium, Bermuda, Bolivia, Brazil, British Virgin Islan, Canada, Chile, China, Colombia, Costa Rica, Czech Republic, Denmark, Dominican Repub, Ecuador, Egypt, England, Finland, France, and Germany. The "Argentina" node is expanded, showing its sub-nodes: Aruba, Australia, Austria, Bahamas, Bangladesh, Barbados, Belgium, Bermuda, Bolivia, Brazil, British Virgin Islan, Canada, Chile, China, Colombia, Costa Rica, Czech Republic, Denmark, Dominican Repub, Ecuador, Egypt, England, Finland, France, and Germany. The "MainReport" tab is selected in the top navigation bar. The main report area contains a table with four columns: "7 Bikes For", "Against The", and "AIC". The table data is as follows:

	7 Bikes For	Against The	AIC
Argentina	Mendoza	\$0.00	\$0.00
	Total	\$0.00	\$0.00
Aruba	St. George	\$0.00	\$0.00
	Total	\$0.00	\$0.00
Australia	New South Wales	\$0.00	\$0.00
	Queensland	\$0.00	\$0.00
	Tasmania	\$0.00	\$0.00
	Victoria	\$0.00	\$0.00
	Western Australia	\$0.00	\$0.00
	Total	\$0.00	\$0.00
Austria	Salzkammergut	\$0.00	\$0.00

Current Page No: 1 Total Page No: 9+ Zoom Factor: 100%

Figure 3-24

Form1

The screenshot shows a Windows application window titled "Form1". The "MainReport" tab is selected in the top navigation bar. The main report area displays a grouped table with four columns: "Total", "\$8,819.55", "\$2,409.46", and "\$5,8". The table data is as follows:

Total	\$8,819.55	\$2,409.46	\$5,8
Argentina	\$0.00	\$0.00	
Mendoza	\$0.00	\$0.00	
Aruba	\$0.00	\$0.00	
St. George	\$0.00	\$0.00	
Australia	\$0.00	\$0.00	
New South Wales	\$0.00	\$0.00	
Queensland	\$0.00	\$0.00	
Tasmania	\$0.00	\$0.00	
Victoria	\$0.00	\$0.00	
Western Australia	\$0.00	\$0.00	
Austria	\$0.00	\$0.00	
Salzkammergut	\$0.00	\$0.00	

Current Page No: 1 Total Page No: 1+ Zoom Factor: 100%

Figure 3-25

Formatting Cross-Tabs

Controlling the column and column sizes within a cross-tab is accomplished by changing the field widths and heights of the objects within the columns or rows. To change the field sizes within your cross-tab, locate the field you want to change and click to select it. This should put a single handle on each side of the field object; you can resize the field by dragging these blue handles. As you resize one field in a column or row, the entire column or row is resized.

Crystal Reports includes a number of preformatted styles that affect the formatting, font size, color, and so on. You can use these to give your cross-tab a standard look and feel. Even if you don't care for any of the styles available, they are often a good starting point for creating your own custom formatting. To apply a preformatted style to your cross-tab object, insert or edit an existing cross-tab and, in the Format Cross-Tab dialog, click the Style tab to open the dialog shown in Figure 3-26.

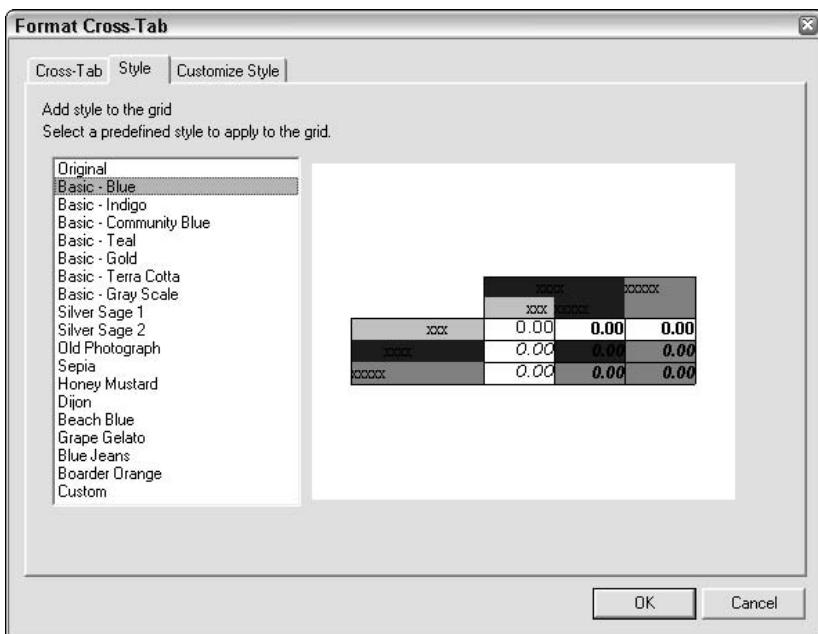


Figure 3-26

You can click to select a predefined style to apply to your cross-tab. As you click a style, the preview window on the right will display a sample of that style. When you are finished selecting your style, click OK to accept your changes and return to your report's design.

Cross-tab styles can be the starting point for creating your own cross-tab format. To modify a cross-tab style, click the Customize Style tab to open the dialog shown in Figure 3-27.

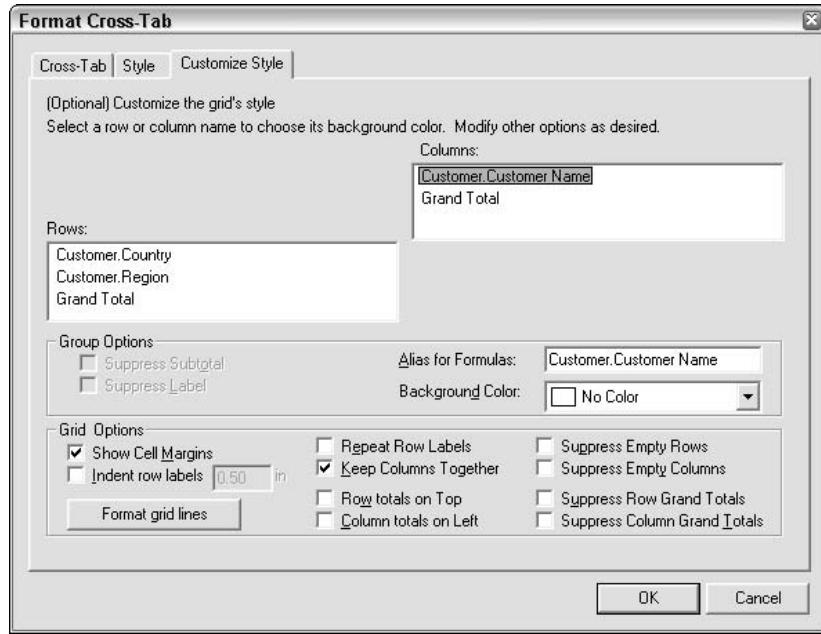


Figure 3-27

Keep in mind that the changes you make here are for a specific cross-tab and do not change the underlying style within Crystal Reports. The options for customizing a particular style are divided into two categories: *group options* and *grid options*.

Group options include the ability to suppress a subtotal or label and change the background color of a particular row or column by highlighting the field and using the drop-down list to select a color. Another group option is the ability to set an alias for formula fields by highlighting a field and entering a new name in the dialog shown in Figure 3-28.

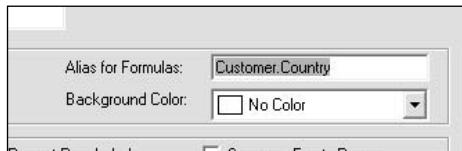


Figure 3-28

In addition to applying and modifying styles to your cross-tab, you also have complete control over the grid options and grid lines that appear in and around your cross-tab, as shown in Figure 3-29.

When you are finished setting the grid lines for your cross-tab or OLAP grid, click OK to return to your report design. The changes you have made should be reflected in your cross-tab. Figure 3-30 shows an example of a cross-tab that has been formatted using the grid settings.

Chapter 3

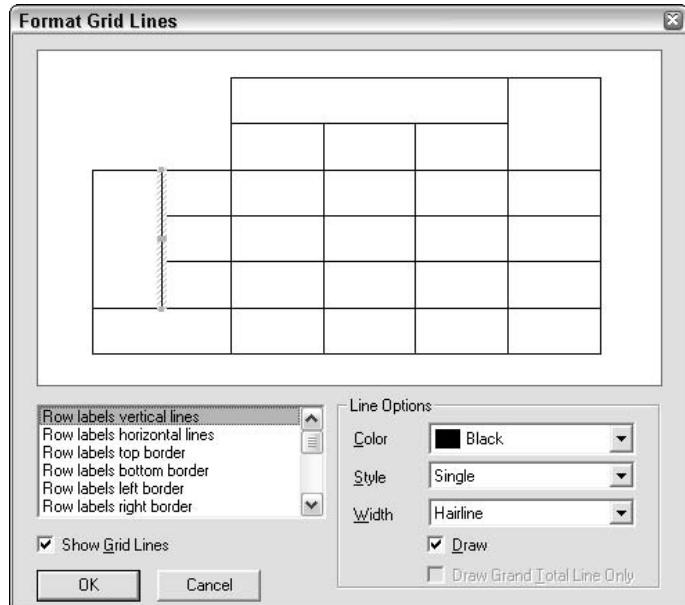


Figure 3-29

The screenshot shows a report viewer window titled 'Form1' with a tab labeled 'MainReport'. The main area displays a table with five columns: '7 Bikes For 7 Brothers', 'Against The Wind Bikes', 'AIC Childrens', 'Alley Cat Cycles', and 'Ankara Bike Company'. The rows represent different regions: 'Total', 'Argentina', 'Mendoza', 'Aruba', 'St. George', 'Australia', 'New South', 'Queensland', 'Tasmania', 'Victoria', 'Western', and 'Austria'. The 'Total' row shows values: \$8,819.55, \$2,409.46, \$5,879.70, \$9,249,042.82, and \$43,000.00 respectively. All other region rows show \$0.00 for all four columns. At the bottom of the table, there are navigation arrows and a status bar with 'Current Page No: 1', 'Total Page No: 1+', and 'Zoom Factor: 75%'.

	7 Bikes For 7 Brothers	Against The Wind Bikes	AIC Childrens	Alley Cat Cycles	Ankara Bike Company
Total	\$8,819.55	\$2,409.46	\$5,879.70	\$9,249,042.82	\$43,000.00
Argentina	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Mendoza	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Aruba	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
St. George	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Australia	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
New South	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Queensland	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Tasmania	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Victoria	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Western	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
Austria	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

Figure 3-30

Working with Charts

Crystal Reports .NET utilizes a sophisticated graphing engine based on technology from ThreeD Graphics and can create just about any type of chart or graph you can imagine.

Throughout this chapter, you have been working on the international sales report. There are groups for the Country, Region, and City fields in the report, and with all of the summary fields and other content, the report runs over multiple pages, making it difficult to visualize the information contained within. In the following pages, we are going to add a chart to the first page of the report we have been working with, which will provide the user with an overview of the information displayed.

To begin, open the report we originally created (and that you copied earlier) in the Report Designer by double-clicking the report in the Solution Explorer.

To add a chart to this report, right-click the report and select Insert → Chart, which will open the Chart Expert dialog, shown in Figure 3-31.

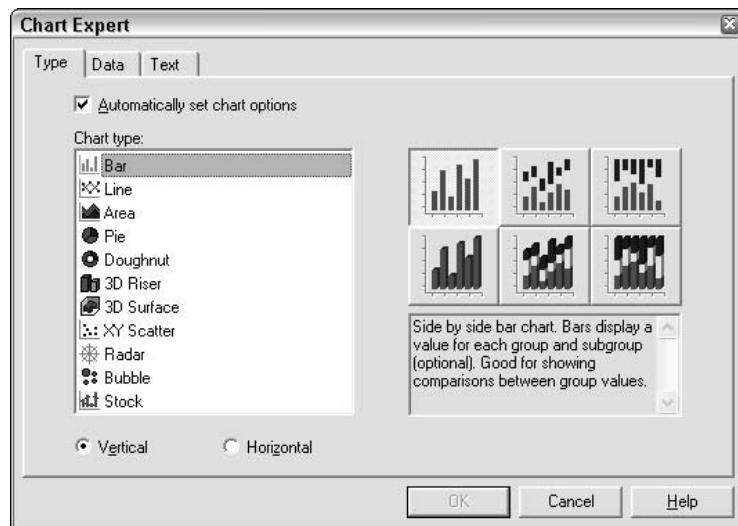


Figure 3-31

You may have noticed when working with the Standard Expert that there was a tab for Charts; the same Chart Expert is used in both places.

There are a number of different templates, including bar charts, pie charts, and 3-D charts, many of which won't be covered in too much detail here. A sample icon of how each will look has been included for you in the Chart Type to the left of the Chart Expert. Please keep in mind that each one of these graph types requires a specific set of information. For example, a pie chart may need only two values passed to it (for the names of the pie slices and their values), whereas a three-dimensional chart may require three or more values.

Chapter 3

When you select a graph type that does not match the data you have available and attempt to exit the report expert, a dialog will appear with three options:

- Continue with Selected Data and Chart Type
- Change Data or Chart Type Selection (Return to Chart Expert)
- Let the Expert Choose the Most Appropriate Chart for Data Selected

If you select the first option and choose to ignore the warning message, your chart will be unpredictable at best. The charting engine will attempt to work with the values you have presented, but it may present a blank graph or one in which the values are just plain wrong. It is better to use the second or third option to ensure that your graph format matches the data available in your report.

In any case, we know that our report has multiple groupings and summary fields, so we should be fine if we select the Pie Chart with the 3D effect by clicking the icon for this graph, shown in Figure 3-32.

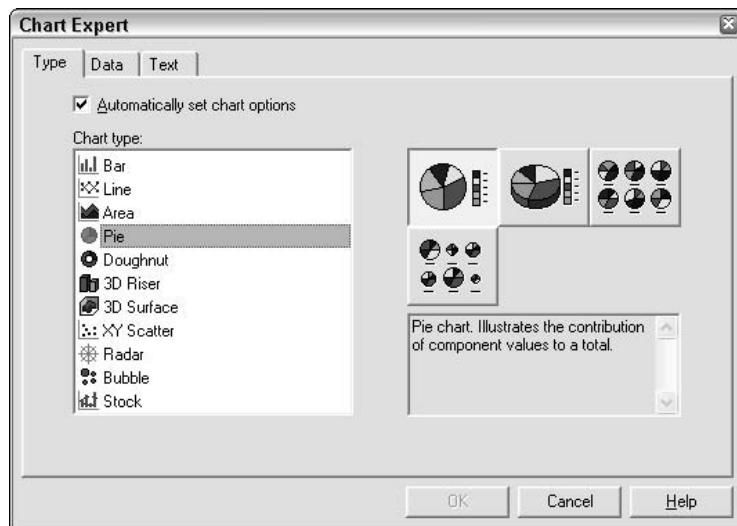


Figure 3-32

Next, select the type of graph you want to create using the Data tab of the Chart Expert, as shown in Figure 3-33.

This dialog says there are four different types to choose from (Advanced, Group, Cross-Tab, and OLAP), but OLAP graphs rely on an underlying OLAP grid inserted onto your report. That feature is not supported in Crystal Reports .NET (at the time of writing, it is supported in the full retail versions of Crystal Reports), so this option will remain permanently grayed out. The Cross-Tab button will also be grayed out if you haven't inserted a Cross-Tab into your report.

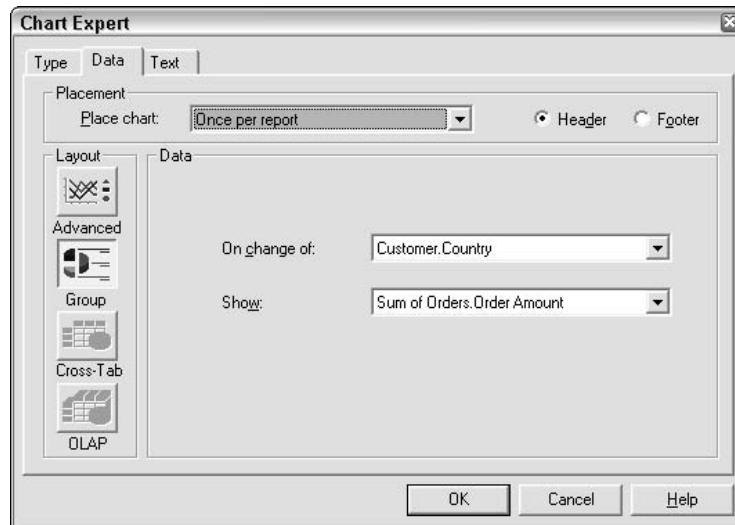


Figure 3-33

Here are the graph types available for use:

- ❑ **Advanced Graphs**—Advanced graphs require two fields: an *On change of* field and a *Show values* field. These are similar to the X and Y fields you would normally use if you were plotting a graph by hand. Just like a manual graph, you can have multiple On change of and Show values fields. You can also determine the order for the On change of field and apply TopN/BottomN/Sort All to the same. For the Show field, you can set the summary operator—sum or average, for example—or choose not to summarize the values at all.
- ❑ **Group Graphs**—These are based on at least one group and one summary field that appear in your report. Group graphs can be placed at the Report Header/Footer level or on the Group Header/Footer level, where they will show the data only for that particular group.
- ❑ **Cross-Tab Graphs**—These rely on an underlying Cross-Tab grid to provide the required data. These types of graphs can also appear on the Report Header/Footer level or on the Group Header/Footer level with the same filtering of data specifically for that group.

In this case, we have a group and a summary field inserted into our report, so you can select to place the chart once per report, specifying On Change Of Customer.Customer Name, and showing Sum of Orders.Order Amount.

Once you have selected the graph and data type for your report, the only thing left in the Chart Expert is to set the text that will appear on your chart or graph, so select the Text tab. By default, Crystal Reports .NET will automatically enter some text for you, or you can override the text and its formatting if you choose. To enter your own text, uncheck the box beside any of the titles and enter your own title in the textbox provided.

Chapter 3

When you click OK to exit the Chart Expert, your Chart will be added to your Report Header. When you see the report in the designer, a pie chart should now be displayed in the Report Header. This isn't an accurate drawing of your graph. In fact it is nothing like your graph; it is just a placeholder to show where in the report the graph will be positioned. If you preview the graph in a Windows form, it will look something like Figure 3-34.

Another option to increase readability would be to make the chart larger. The chart object inserted into your report is just like any other object in that it can be resized to fit your needs; you could even resize the graph to take up the entire first page if that suits the needs of the report.

If you have a graph or chart inserted into your report, you can control the content by right-clicking directly on top of the chart and selecting an option from the menu. The Chart Expert can be used to alter the graph in the way that we have described so far in this section. You may have noticed that the Chart Expert includes an option on the first tab labeled "Automatically set chart options." If you uncheck this box, two more tabs will appear, allowing you to control other aspects of the report, including the formatting for the graph axes and the general settings (like color, marker size, and so on).

Under Format Chart are the Template, General, and Titles options, which also allow you to customize the appearance of your chart. The options in all of these menus are context sensitive; for example, you can perform different types of formatting to a pie chart than you can to a bar chart. These options are all relatively simple, so we'll leave it to you to explore the almost endless possibilities.

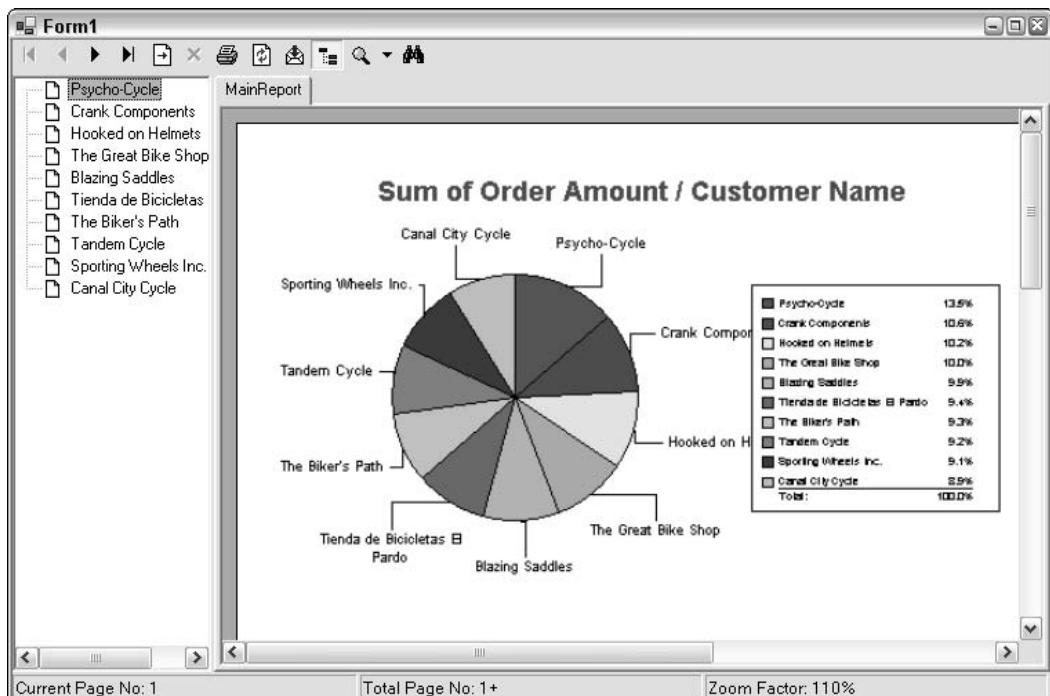


Figure 3-34

Unfortunately, if you have used the retail version of Crystal Reports before, you will probably be wondering what happened to the Chart Analyzer, which allows you to open the graph in another tabbed window. Crystal Reports .NET does not include the full capabilities of the Chart Analyzer, so if you really need to use some of the advanced formatting features for charts and graphs, you are going to have to buy a retail copy of Crystal Reports.

Working with Subreports

Within Crystal Reports .NET, multiple subreports can be combined into one main report, which allows you to create information-rich reports from multiple sources and display this information side by side.

Both linked and unlinked subreports are available in the code download, found in the solution: C:\CrystalReports\Chapter02\Demo\Chapter2.sln. Open and run this solution from Visual Studio .NET and choose the report from the drop-down box at the top right of the window.

To run the unlinked report, you will need access to the Northwind sample database.

Subreports come in two varieties—unlinked and linked. *Unlinked subreports* allow you to insert subreports that are totally unrelated to the main report content. In Figure 3-35, a sales graph has been inserted into an employee listing report (included in the sample files as employee_listing_unlinked.rpt). Both of these reports were developed independently and are from different tables and a different data source.

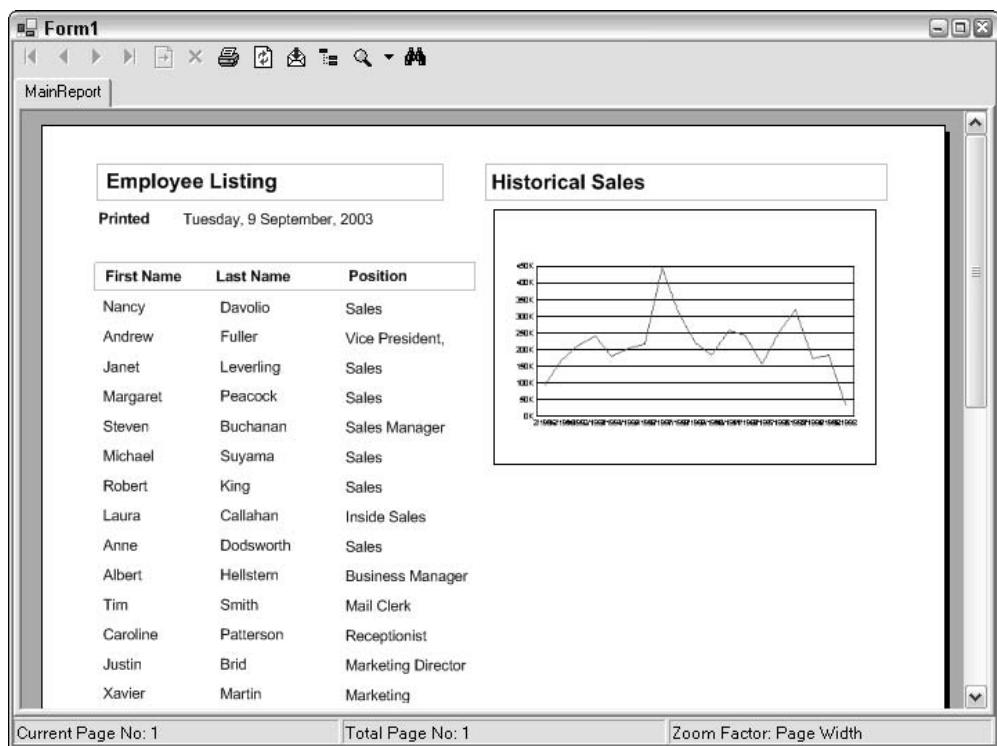


Figure 3-35

Chapter 3

The second type of report is the *linked subreport*. Linked subreports allow the passing of parameters and variables between the main report and the subreport, which can be used to filter the subreport content. The report shown in Figure 3-36 has a main report that is the same employee listing report, only this time a linked subreport has been inserted showing a commission amount for each employee (included in the sample files as `employee_listing_linked.rpt`). The commission report is a separate report, but it is inserted into the details section and linked on the employee ID field.

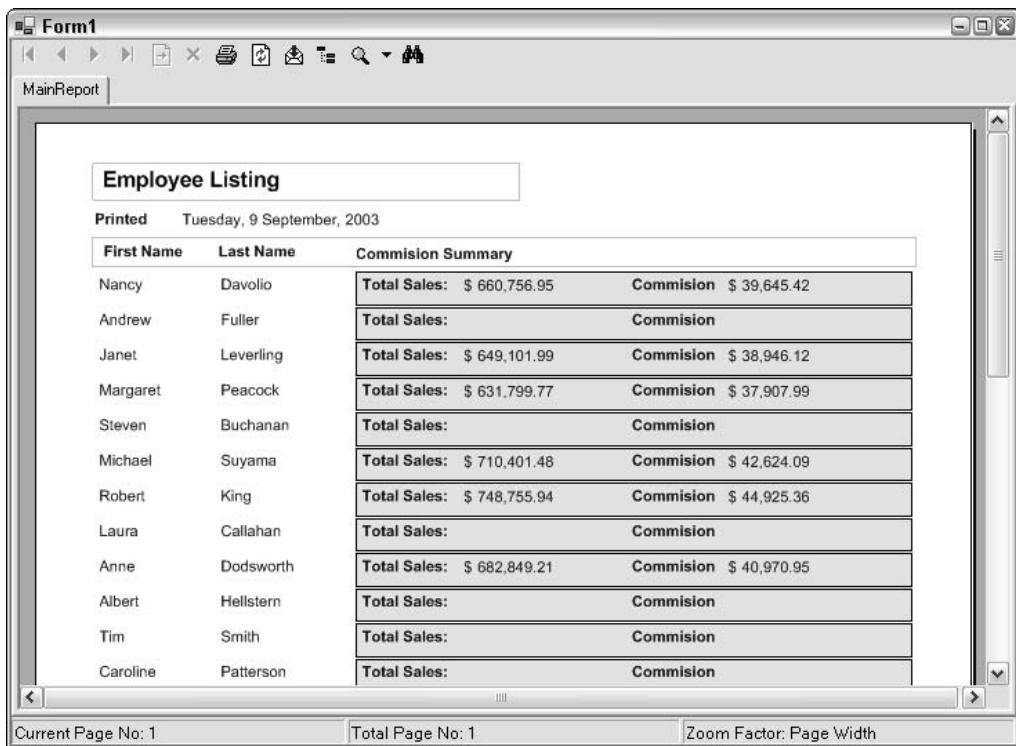


Figure 3-36

For each employee, the subreport is run again, and the employee ID is used in record selection on the subreport. When the page is printed, each instance of the subreport is printed next to the corresponding employee, with only their details shown.

Inserting Subreports

To insert a subreport into your report, right-click a blank area of the report, and select Insert → Subreport from the right-click menu, which will open the dialog shown in Figure 3-37.

You have three options for adding a new subreport: First, if the report you want to add is a report that exists in your current project, you can use the drop-down list at the top of the dialog to select the report.

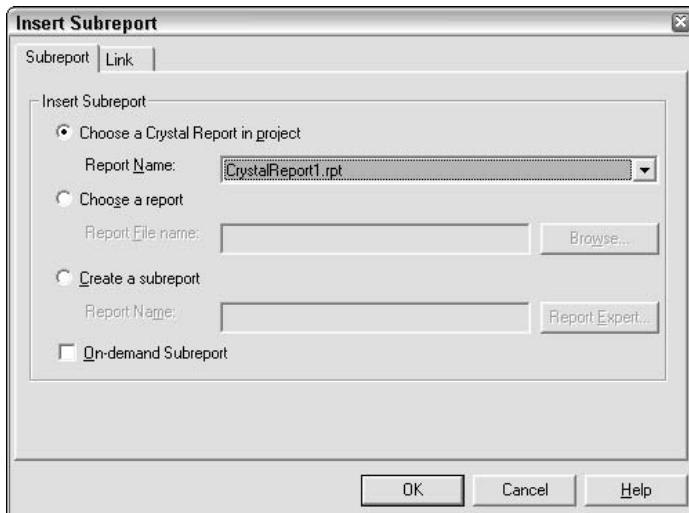


Figure 3-37

Second, if you want to add an existing report that has been saved outside of your project, use the Choose a report option and the Browse button to locate the report you want to add as a subreport.

And finally, if you want to create a subreport from scratch, select the Create a subreport option, enter a name for the subreport, and click the Report Expert button to launch the Standard Expert, which we looked at in Chapter 2, to create your report. The Standard Expert is shown in Figure 3-38.

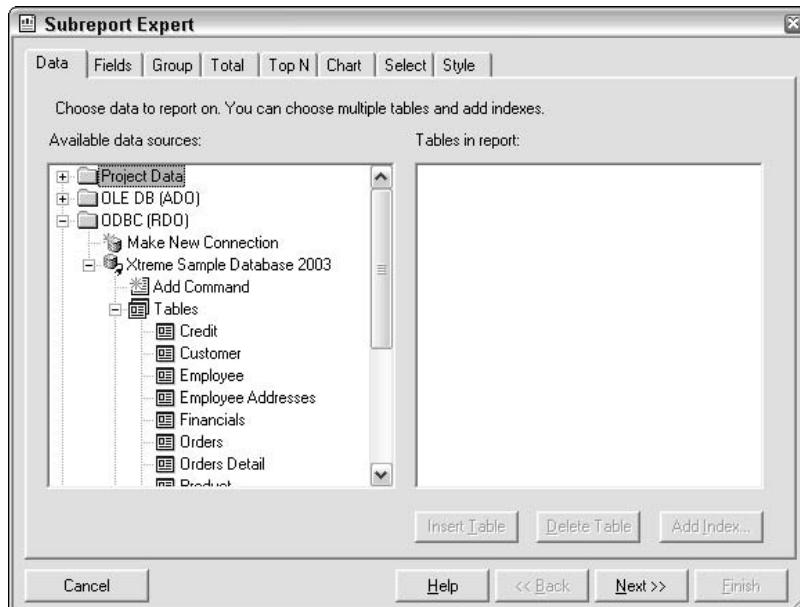


Figure 3-38

Chapter 3

Once you have selected or created your report, click the Links tab to open the dialog shown in Figure 3-39.

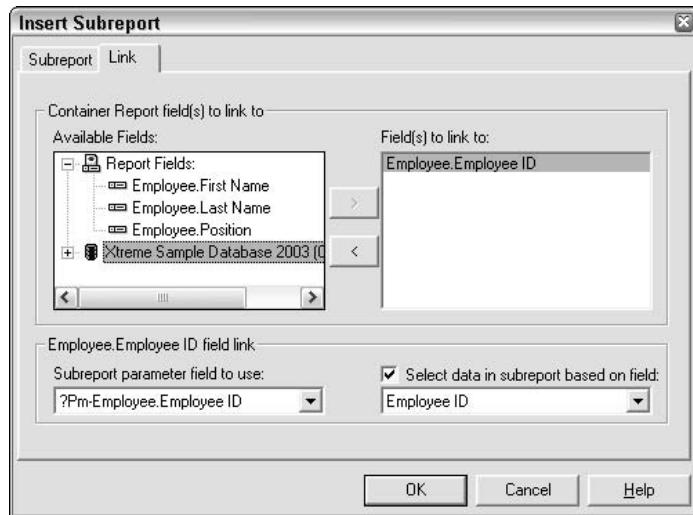


Figure 3-39

Using this dialog, you can create a linked subreport, where the subreport details will be filtered by a field in your main report. To select a field from the main report, highlight the field and click the right-arrow to add it to the list of selected fields. Then, using the drop-down lists at the bottom of the page, select the corresponding field in your subreport.

Remember, you don't actually have to specify any links at all, but if you do not, the subreport will not be filtered—that is, it will run for all records in the subreport, wherever you place it. If you place an unlinked subreport in the details section of your main report, for example, it will run once for every detail record.

When you are finished, click OK to return to the report designer; your subreport will be attached to the tip of your cursor, and you can click to place it on your report. It will appear as a box with a border around it and a label for the subreport name.

Changing Subreports

To change the subreport name that appears when you preview your report, right-click the subreport and select Format from the right-click menu. Using the option on the Subreport property page, you can change the Subreport Preview Caption by clicking the X+2 button and entering a new name enclosed in double-quotes. Whenever the subreport name is shown (in the main report, in the tool text, on a design tab for the subreport, and so on), this name will be used.

Another common format change is the subreport border; Crystal Reports places a border around any subreports you have inserted into a main report, and this is usually the first default formatting option you will want to turn off. To change the border around a subreport, right-click the subreport and select Format. From the Border tab in the Format Editor dialog, shown in Figure 3-40, you can change all four of the Line Style drop-down boxes from Single to None.

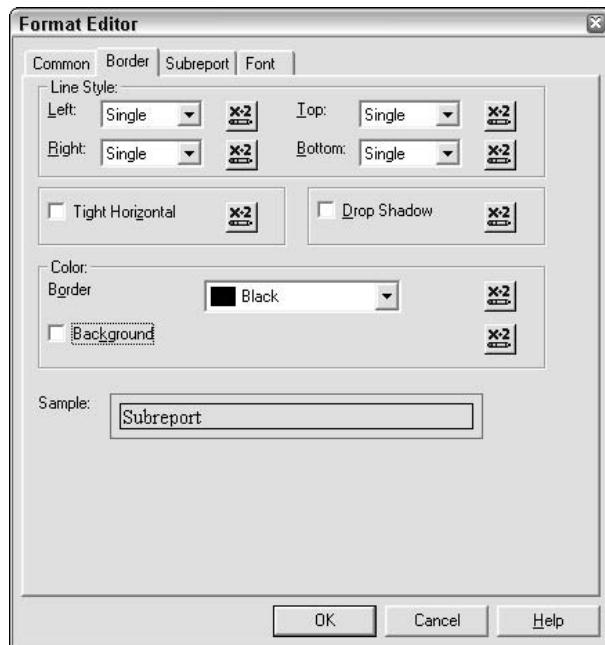


Figure 3-40

Alternatively, if you want a border around your subreport, you can use the drop-down boxes and options to select a line style (Single, Double, Dashed, Dotted) and color, as well a background color and drop shadow. When you are finished editing the borders and colors for your subreport, click OK to return to your report design.

Subreport links are usually set up when you first insert a subreport, but you can change subreport link-age as your needs and report structure change. You will need to locate the subreport you want to change, right-click directly on top of it, and then select Change Subreport Links from the right-click menu. This will open the Subreport Links dialog and allow you to change the links to your subreports.

Creating On-Demand Subreports

On-demand subreports have become a popular option with Crystal Reports developers; an on-demand subreport included within a main report is not processed until required. Such a subreport can appear as a link or even an element of a main report, but the subreport is not actually processed until the user clicks the link or element, like the one shown in Figure 3-41.

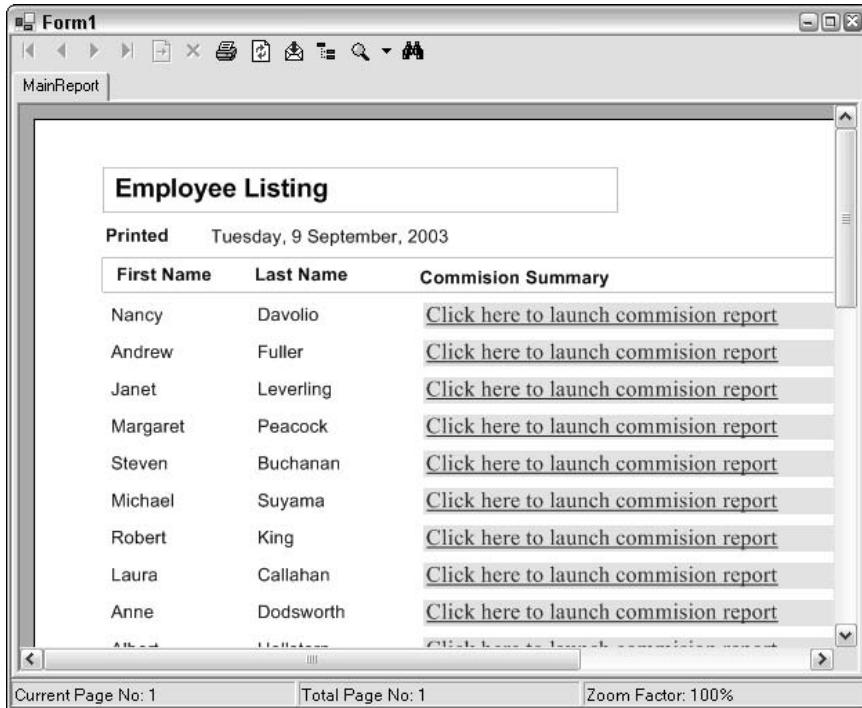


Figure 3-41

When the subreport is processed, it is opened in a separate preview tab and can be viewed and printed independently. This is a great way to ensure efficient reporting; the details contained within a subreport are not returned until a user requests them.

To create an on-demand subreport, you can insert either a linked or unlinked subreport. Once the subreport has been inserted onto your report, you will need to set some options for on-demand reports. Locate the subreport you want processed on demand, right-click directly on top of it, select Format, and then select the Subreport tab, which will open the dialog shown in Figure 3-42.

There is only one setting required to process a subreport on demand, and that is the check box shown at the top of the dialog. With this option enabled, a subreport will not be processed until the user clicks it.

The user may need a little prompting to understand what is going on. For this purpose, you can select On-demand Subreport Caption by clicking the X+2 button. Using the Crystal Reports Formula Editor, enter a caption for your on-demand subreport and enclose it in quotation marks. This text is what the end user will see on the on-demand subreport link when previewing your main report. When you are finished, click the Save and Close button in the upper left corner to exit the formula editor (see Figure 3-43).

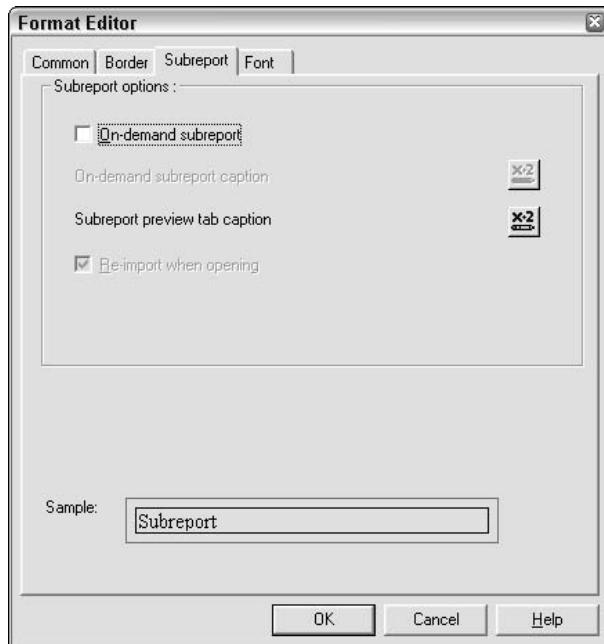


Figure 3-42

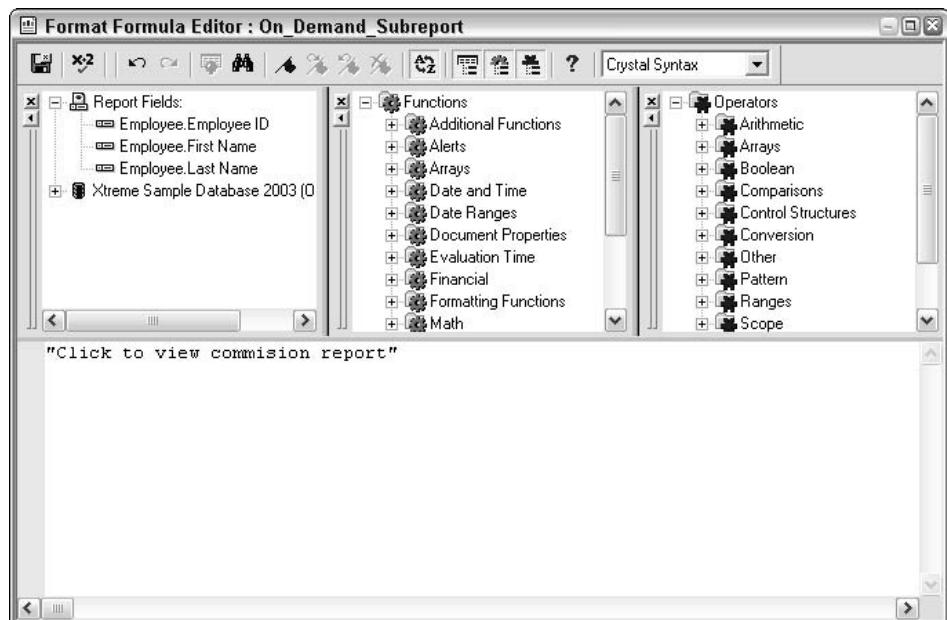


Figure 3-43

Saving and Reimporting Subreports

Subreports can be saved to a separate, independent report file and can also be reimported when the main report is opened. Saving subreports to an external file allows you to break up main reports and subreports so users can print the subreport independently or use it in other reports. To save a subreport, right-click directly on top of the subreport object in your main report and select Save Subreport As. Specify a file name and click OK, and your subreport will be saved to a separate file.

Reimporting subreports provides a creative way of using them. Using this facility, you can create a number of reports that can serve both as subreports and as reports in their own right. This setting is available for linked or unlinked subreports that have been inserted from an existing report file. To enable this setting, locate a subreport that has been inserted from an existing file, right-click directly on top of the sub-report object, and select Format from the right-click menu.

The Subreport property page contains a Re-import When Opening check box. Enable this option, as shown in Figure 3-44, and Crystal Reports will look in the last file location and attempt to reimport the report file you originally used.

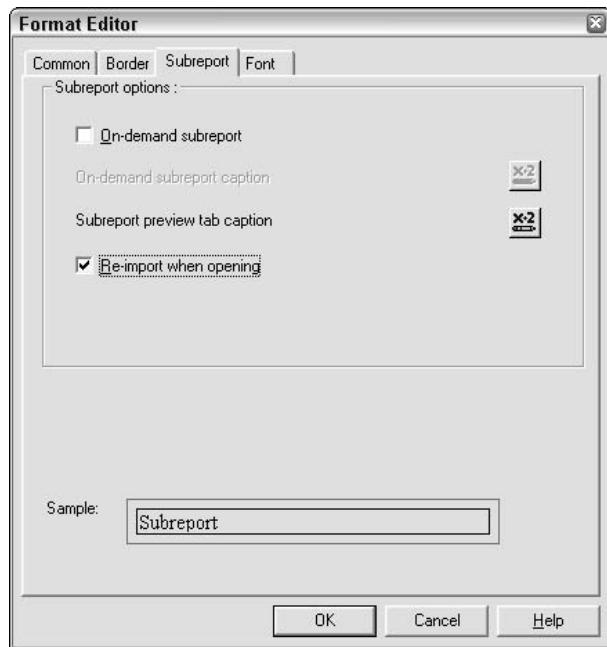


Figure 3-44

If the report file name or location has changed, Crystal Reports will display the error message and allow you to select the correct file name or location for the subreport you want to import.

You can also force reimport at any time by right-clicking the subreport object and selecting Re-import Subreport from the right-click menu.

Working with Parameter Fields

Parameter fields enable you to create reports that could be used in a variety of ways, prompting the user for all kinds of information, including values to be used with record selection, sort orders, report titles, comments, and more. Parameter fields also give you a quick and easy way to create reports that can serve many users and purposes.

Parameter fields can be used in your report in a number of different ways. The simplest use of a parameter field is when you want to display some text on your report. This could be the report title, a brief explanation of the report, your name, or just about anything you could imagine where you need to add some text to your report at runtime.

A second use for parameter fields is in conjunction with formulas: You may want to prompt the user for the sales tax amount, which will then be calculated; a particular type of shipping (two-day, overnight, and so on), which could then be used to calculate shipping cost; or even a discount amount when printing invoices.

The third and most popular use of parameter fields has to be record selection. By using parameter fields, you can create a single report that can be sliced many different ways, depending on what values the user enters for the parameter fields. These fields are then used in the record selection formula to determine what data is brought back from the database.

Creating a Parameter Field

The first step in creating a parameter field is specifying a name for that field. Once you have created your parameter field, Crystal Reports will enclose this field name in curly brackets, preceded by a question mark to indicate it is a parameter field (`{?ParamFieldName}`). You can also enter prompting text that will appear whenever the Parameter Field dialog appears, as shown in Figure 3-45. Prompting text should help the user understand what to enter in the value field of the dialog (for example, "Please enter a state for this report.").

By default, any parameter you create will have a default type of String, but there are actually seven different field types you can use, including:

String	For entering alphanumeric text
Currency	For prompting users to enter an integer with two decimal places
Date	Used to enter a standard date, in the format Month/Day/Year
Date Time	For prompting for a date/time string, in the format Month/Day/Year Hour/Minute/Second/AM
Time	For entering the time in the format Hour/Minute/Second/AM
Number	Used to enter a number with variable decimal places
Boolean	Used to prompt users for a true or false response

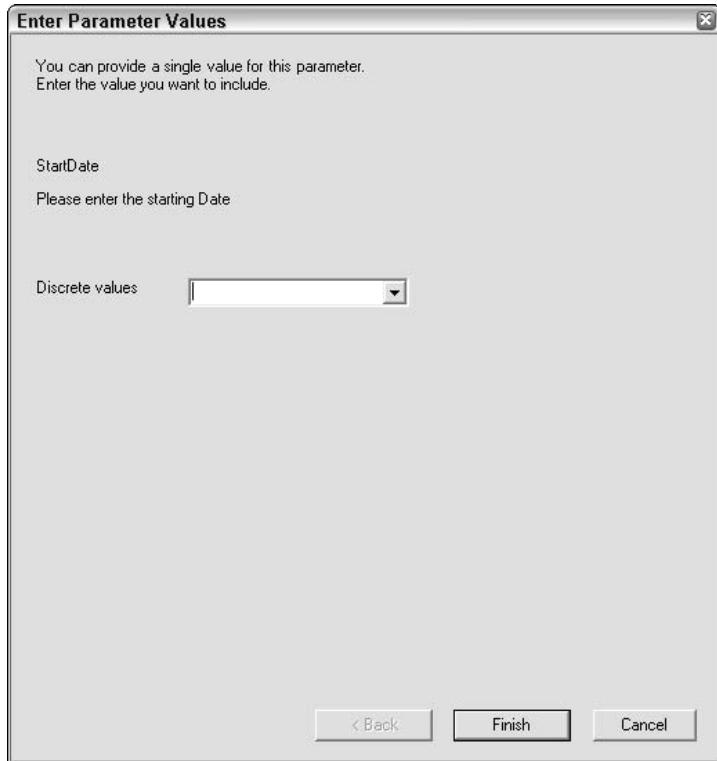


Figure 3-45

Which field type you choose depends on how you are going to use the field in your report. Other attributes that can be set when creating a parameter field include:

- Allow Multiple values**—Allows you to enter a list of values for your parameter field
- Discrete**—Allows you to enter a single value
- Range**—Allows you to specify an inclusive range, using a start and value
- Discrete and Range**—Allows a combination of the previous two attributes

Most of the attributes discussed here are optional; to create a parameter field, the only requirement is that you give your parameter field a name and choose a field type.

To create a simple parameter field to be used in your report, open the Field Explorer and right-click the Parameter Fields node and select New from the right-click menu. Using the dialog shown in Figure 3-46, type a name for your parameter field. In this example, we have named the parameter field “EnterState.”

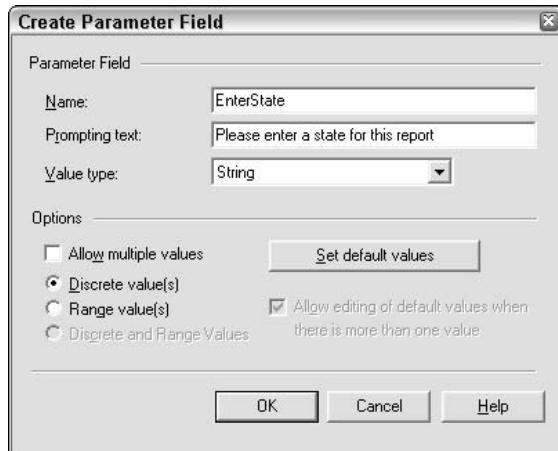


Figure 3-46

Once you have dragged a parameter field onto your report, you will be prompted to enter a value for the parameter the next time the report is previewed using the Windows report viewer, which we will look at in the next chapter, and the generic parameter prompting dialog, as seen in another example in Figure 3-47.

Next, enter any text you wish to appear when the user is prompted for information (for example, “Please enter an employee ID for this report.”). Using the combo box labeled “Value type,” select a data type for your parameter field.

Click OK to accept your changes. Your parameter field should now appear within the Field Explorer, ready to be used on your report. To insert a parameter field you have created, you can simply drag it from the Field Explorer onto your report design.

If you are using parameters with a Web application, you will need to set the parameter before you view the report. Otherwise you will receive an error message.

After you have entered a value for your parameter field, that value will be displayed in the report preview until you refresh your report and specify you want to prompt for a new parameter value.



Figure 3-47

Optimizing Report Performance

If you have worked with reporting applications before, there is usually a bit of time spent on optimizing report performance; users are not happy with reports that run for three minutes, let alone three hours. Over the years, there have been significant enhancements within the Crystal Print Engine that have improved performance and cut down on processing time, but the majority of poor report performance does not lie within the Report Designer but rather in how the report is designed and in the underlying data.

For example, if you have a report that has been developed across an Oracle table that contains 500,000 rows of data, the report is going to take a while to run, regardless of whether you are using Crystal Reports .NET or just submitting an SQL Statement from a PL/SQL command prompt. If you believe that a report's performance could be improved, ask your DBA or architect to review the tables, views, and other data impedimenta that you are using in order to verify that you have used the correct primary and foreign keys and that you have taken the most direct route to join tables in your report.

Following from that, you also may want to take the SQL that Crystal Reports .NET generates (right-click your report and select Database → Show SQL Query) and paste it into SQL*Plus Query Analyzer or any other SQL query tool supported by your database platform to see how long it runs. Your DBA could also provide suggestions on ways to improve the SQL statement generated, and you can then use the optimized SQL as the basis for your report.

With Crystal Reports .NET itself, there are a couple of options that can help with performance. To view these options, right-click your report and select Report → Report Options.

There are two options in this dialog that can aid with performance: The first, Use Indexes or Server For Speed, for use with databases that use indexes, will use the index on the database server to sort and retrieve records faster than if the index was not used. The second option, Perform Grouping On Server, for reports that have groups, will push the grouping back to the database server, provided that the details within your report are suppressed. (We can't show detailed records in the report because it actually changes the SQL statement and uses a GROUP BY clause.)

Summary

This chapter ran through the laundry list of Crystal Reports features and functionality without even stopping to take a breath! Most report designers will not use all of these features in a single report, but hopefully this chapter will serve as a reference that you can turn to as you need to add different features to your own reports.

So with a bit of report design under your belt, it's time to take a look at actually integrating the reports you have been creating with your own applications. In the next chapter we'll pick up with that topic, starting with integration for Windows applications.

4

Report Integration for Windows-Based Applications

With a bit of basic report design under our belts, it's time to look at actually integrating reports into your Windows-based applications. In the previous chapter, we looked at how to create reports using some of the features within Crystal Reports .NET.

In this chapter, we are going to look at how to integrate and view those same reports from Windows applications and how to customize our reports at run time using the rich object models provided. We will cover:

- Determining the correct object model
- The `CrystalDecisions.Windows.Forms` namespace
- Using the Crystal Windows Forms Viewer
- Customizing the Windows Forms Viewer
- Passing information to the Windows Forms Viewer

Throughout the chapter, we will be looking at code examples to illustrate the use of various features, and by the end of the chapter, you should be familiar with the majority of report integration concepts and be ready to apply them to your own application development.

Obtaining the Sample Files

All of the projects we will work through in this chapter (like the custom viewer shown in Figure 4-1), as well as the associated sample reports are available for download from www.wrox.com.

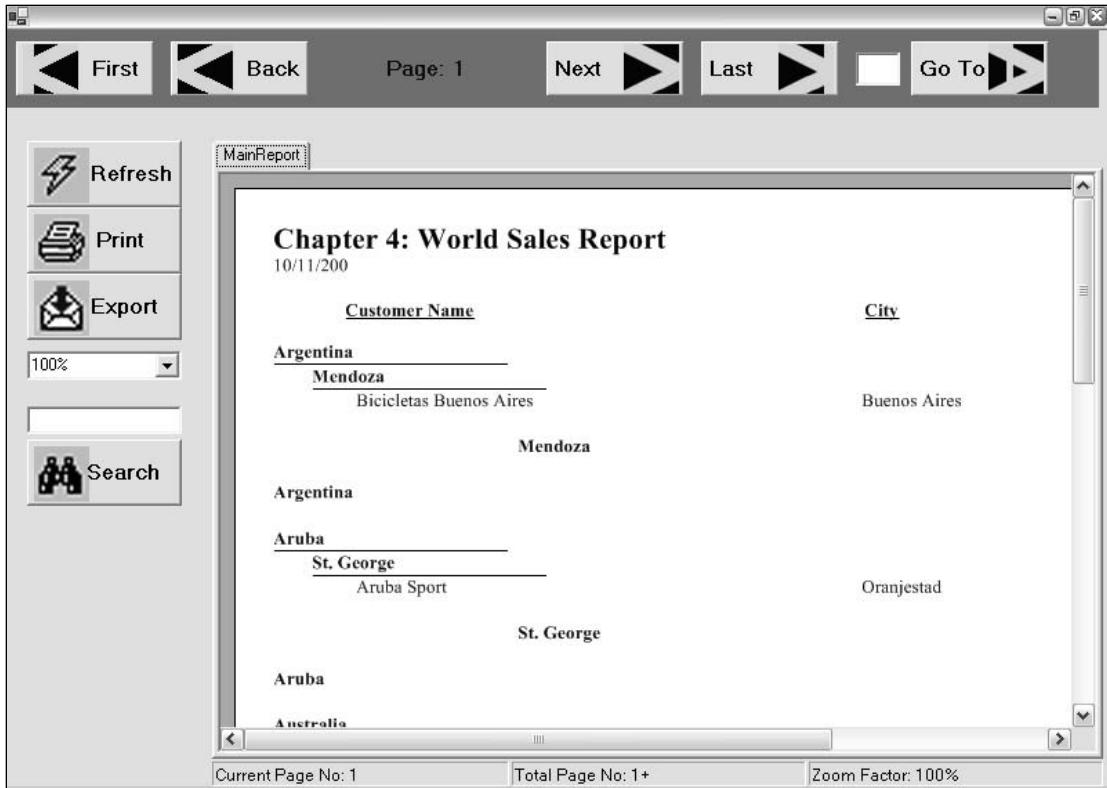


Figure 4-1

As you go through the chapter, you will be able to actually look at the application we are creating and other examples that illustrate points along the way.

Planning Your Application

Integrating reports is an easy way to add value to your application and can be an important component of your application's offering. Before we get started on actually integrating Crystal Reports .NET with your Windows application, we need to do a little planning to make sure the integration goes smoothly.

First, we need to have a report (or suite of reports) to work with. In Chapter 2, "Getting Started with Crystal Reports .NET," we walked through the report planning and design process, so you should have a start on the skills you need to design reports.

If you flipped straight to this chapter or haven't gotten into designing your own reports yet, there are some sample reports located in C:\program files\Visual Studio .Net 2003\Crystal Reports\Samples\ that you can use to practice concepts from this chapter; alternatively, there are the sample reports included in the download file as well.

We need to plan for how those reports will be delivered to users and the forms that will be required to host them. Crystal Reports .NET uses a feature-rich report viewer that can be inserted onto a Windows Form and used to view reports. The viewer itself has an extensive object model, allowing you to set the source of the report, the appearance of the viewer itself, and what happens when different events fire.

Most applications can utilize a single Windows Form hosting the Crystal Report Viewer and simply pass properties like the report source and viewer settings to this form. This lends itself to a number of creative solutions for user personalization and settings. You could store viewer settings and preferences in a table or XML file for each user (or group or role) and apply these settings when viewing a report.

In addition, you could also set specific record selection formulas for different groups of users, allowing them access only to the data applicable to them. You could also create a custom user interface, allowing users to set and retain parameter settings for future use, or even to keep their printing or export preferences, such as frequently used e-mail addresses.

Ultimately, the report integration should be driven by the user's requirements, but how these features are delivered is up to you. As you go through the rest of the chapter, think about how the different customization features could be used in your development. If you are not at a point where you can integrate these features into your application, all of the properties, methods, and events are grouped together by function to make it easier to come back and look them up.

Exploring the Development Environment

Visual Studio .NET provides a rich integrated design environment for developing Windows and Web applications, and there are a number of components and shortcuts to help us integrate Crystal Reports into Windows applications.

To begin, in the toolbox under the Windows Forms section, you will find the `CrystalReportViewer`, which we will be working with a little later. When you draw this viewer on a Windows Form, you can set a number of properties and use the viewer to display a preview of your report, as shown in Figure 4-2.

In addition to the `CrystalReportViewer`, there is also a `ReportDocument` component available in the components section of the toolbox. We use this component to add *strongly typed* and *untyped* reports to a form, for use with the viewer mentioned earlier. (Don't be too worried about it at the moment, as we'll cover that a little later in the chapter.)

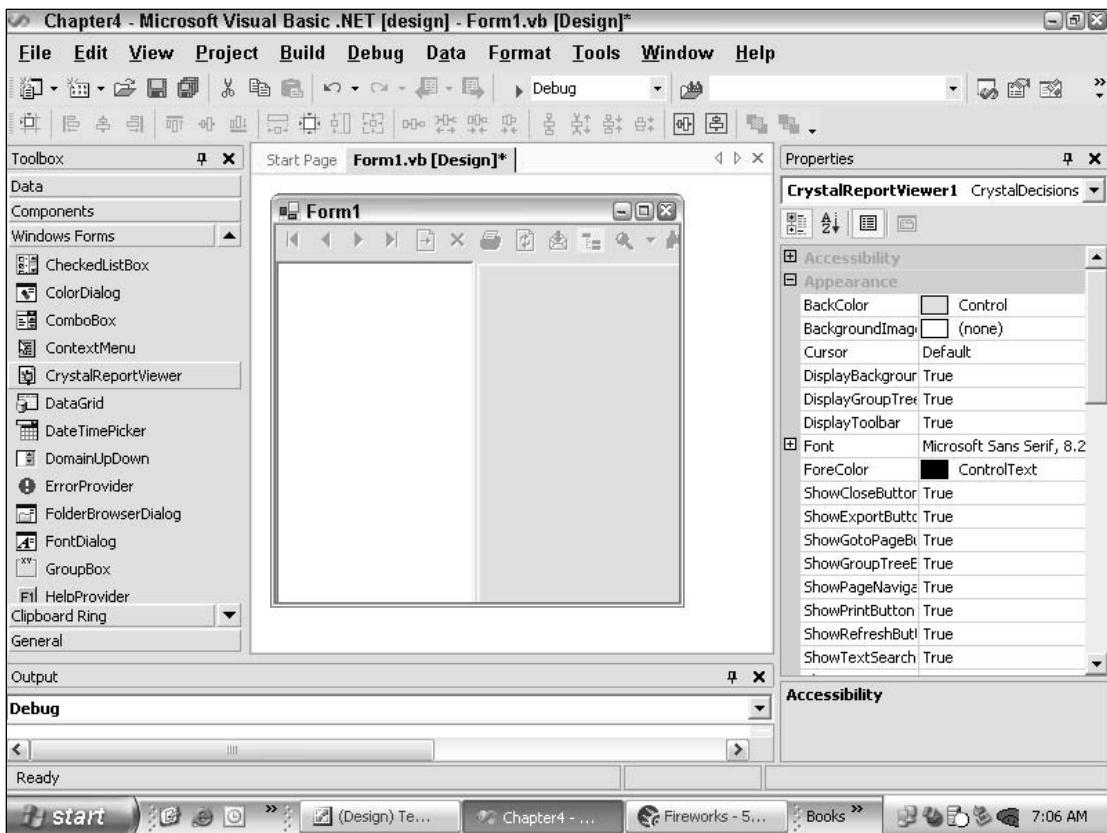


Figure 4-2

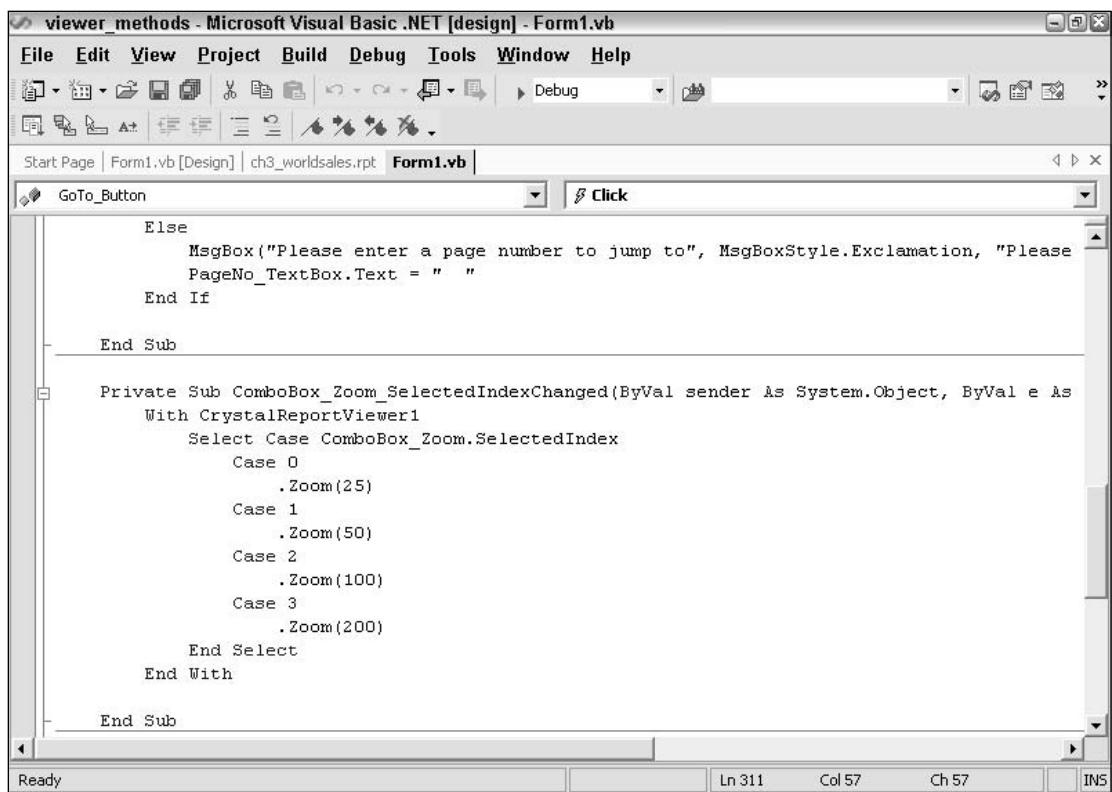
Finally, the majority of our report integration will take place in the code view of the form, as shown in Figure 4-3.

All of the properties, methods, and events related to Crystal Reports object models and integration can be set and modified through this view, as well as through the Properties window.

Starting a New Windows Application with VB .NET

To begin, we need to create a new Windows application using Visual Basic .NET. If you want, follow along using the sample code available for this chapter; you will find a number of projects included that correspond to the sections in this chapter. If you want to get down and get your hands dirty creating your own project as we go along, then from within Visual Studio, select File → New → Project, and from within Visual Basic Projects, select Windows Applications and specify a name, as shown in Figure 4-4 (in the sample code, we have called this project `viewer_basic` and we have saved it to `C:\Crystal .NET2003\Chapter04`)

Report Integration for Windows-Based Applications



The screenshot shows the Microsoft Visual Studio IDE in design mode for a Windows application. The title bar reads "viewer_methods - Microsoft Visual Basic .NET [design] - Form1.vb". The menu bar includes File, Edit, View, Project, Build, Debug, Tools, Window, and Help. The toolbar has various icons for file operations like Open, Save, Print, and Find. The status bar at the bottom shows "Ready", "Ln 311", "Col 57", "Ch 57", and "INS". The main code editor displays the following VB.NET code:

```
Else
    MsgBox("Please enter a page number to jump to", MsgBoxStyle.Exclamation, "Please")
    PageNo_TextBox.Text = " "
End If

End Sub

Private Sub ComboBox_Zoom_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
With CrystalReportViewer1
    Select Case ComboBox_Zoom.SelectedIndex
        Case 0
            .Zoom(25)
        Case 1
            .Zoom(50)
        Case 2
            .Zoom(100)
        Case 3
            .Zoom(200)
    End Select
End With

End Sub
```

Figure 4-3

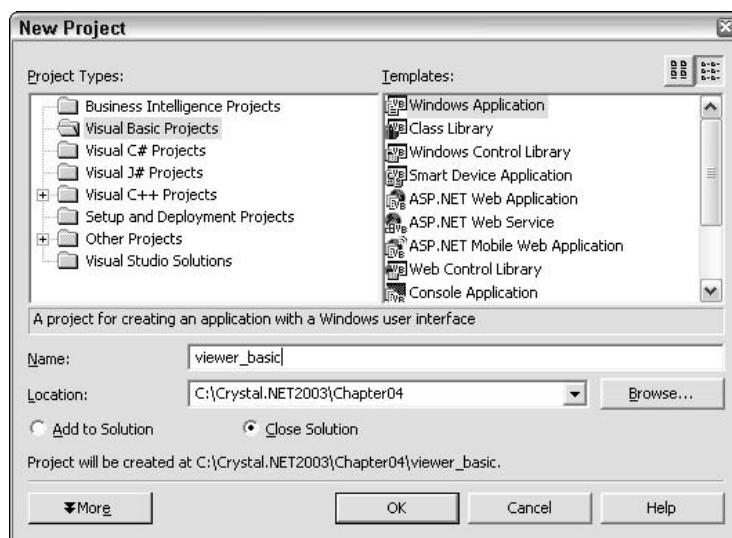


Figure 4-4

Throughout the chapter, we will be using only one or two forms to demonstrate different integration features; your own applications will probably have multiple forms and launch reports from any number of them, but the same concepts can be applied.

Before we actually look at any code, we need to go back to the integration features that you want to incorporate into your application and select the appropriate object model(s).

Determining the Correct Object Model

When working with Windows applications, you have two different object models to choose from, depending on your particular needs. The first, contained within the *Crystal Reports Windows Forms Viewer* object model (`CrystalDecisions.Windows.Forms`), contains all of the functionality required to view a report in the Crystal Reports Windows Forms Viewer. It also includes the ability to set database logon information, pass parameters and record selection, control the viewer's appearance, and view reports, including reports consumed from an XML Report Web Service.

Using this object model, you can satisfy most basic report integration requirements, but you have no control over the report itself; you won't be able to change the record selection for any subreports that appear in your report, and you won't have access to modify report elements, such as groups, and sorting and formula fields.

For more information on working with groups and sorting check out Chapter 3, "Designing Reports."

You also need to use the Crystal Reports Engine object model (`CrystalDecisions.CrystalReports.Engine`) for complete control over your report and the objects and features contained within. Using the Crystal Reports Engine, you are provided with a rich object model that can be used to modify even the smallest elements of your report.

You will also need to use this object model if you are using ADO (.NET or Classic ADO) as the data-source for your report (which is covered in Chapter 7, "Working with .NET Data").

It is important to note that the Crystal Reports Engine object model cannot stand alone; it provides no way to view a report and relies on the Crystal Reports Windows Forms Viewer to actually view the report. The functionality covered by the Report Engine is reviewed in Chapter 9, "Working with the Crystal Reports Engine," as well as examples of some of the most commonly used features.

Crystal Decisions recommends that you do not overlap these two object models and try to use properties and methods from both at the same time. An example would be where you are setting a parameter field value in the Report Engine object model; you wouldn't also want to try to set a parameter field in the same report using the Crystal Reports Windows Forms Viewer object model. Try to pick an object model based on your requirements and stick with it throughout your application.

A good rule of thumb to apply when making a decision about which object model to use is that the Report Viewer can be used with simple applications (preview, print, export) where you don't need to change the report's design or elements within the report. If you need more granular control over the report content, you are going to need to use the Report Engine in conjunction with the Report Viewer.

Understanding the CrystalDecisions. Windows.Forms Namespace

The `CrystalDecisions.Windows.Forms` namespace consists of a number of classes that provide functionality specific to viewing reports. As you look through the classes below, you can easily map each back to some function within the viewer itself. In the section immediately following, we are going to look at each of these classes in depth and learn what can be done with each.

Class	Description
<code>CrystalReportViewer</code>	Contains the properties, methods, and events relating to the <code>CrystalReportViewer</code> and viewing reports.
<code>DrillEventArgs</code> and <code>DrillSubreportEventArgs</code>	Provides data for the <code>Drill</code> and <code>DrillDownSubreport</code> events on main and subreports. Drill events fire when a user drills down into a group or summary on a particular report or subreport.
<code>ExceptionEventArgs</code>	Provides data for the <code>HandleException</code> event. A <code>HandleException</code> event occurs when there is an error or exception when setting the report properties or viewing the report. It is primarily used for troubleshooting and error messages.
<code>NavigateEventArgs</code>	Provides data for the <code>Navigate</code> event. When a user navigates through the pages of a report, the <code>Navigate</code> event fires each time. This can be used to notify the users when they have reached the last page, call custom actions, and so on.
<code>SearchEventArgs</code>	Provides data for the <code>Search</code> event. The <code>CrystalReportViewer</code> includes an integrated search function to search for values within a report. The <code>Search</code> event fires when a user searches for a value, and could be used to trigger a search of another report or other report descriptions.
<code>ViewerEventArgs</code>	Provides data for the <code>Viewer</code> events. The <code>Viewer</code> event fires when some action has occurred within the viewer itself (for instance, when the report has been loaded), and can be used to launch other actions when the viewer loads.

Table continued on following page

Class	Description
ZoomEventArgs	Provides data for the ViewZoom event. The ViewZoom event fires when the zoom is changed on the viewer, and can be used to suggest the best resolution for a particular report. Also, if you are showing two reports in viewers side-by-side, it can be used to synchronize the zoom factor between the two.

Using the Crystal Report Viewer for Windows Forms

For report files that exist externally to your application (for instance, as a standalone report file, created with either this or a previous version of Crystal Reports), there is not much to creating a simple preview form for your report.

In this example, we are going to use the `viewer_basic` project, which you created earlier. In the project, there should be a default `Form1` that we can use. Locate the `CrystalReportViewer` within the toolbox and drag or draw the Report Viewer onto your form.

Once we have added the Report Viewer to our form, we now need to set the `ReportSource` property to point to the existing report file. The sample files for this chapter include a sample report (`ch4_worldsales.rpt`) that we can use. Using the property pages for the Crystal Report Viewer, set the `Report Source` property to the location of this report file. For instance, if you have put the sample files in your own personal directory, enter the full report file name and path, (`C:\CrystalReports\Chapter04\ch4_worldsales.rpt`).

You can now run your application, and when you preview the form, your report will be displayed in the Crystal Reports Viewer, as shown in Figure 4-5.

The viewer interacts with the Crystal Reports Print Engine, runs the report, and displays the results. From your report preview, you can drill-down into the details, search for a value, print, and export (as shown in Figure 4-6) without having to do any additional coding.

If you only have one or two reports that you want to integrate into your application and you don't need to customize any features at run time, this may be all you need. However, for applications that require a more sophisticated integration with Crystal Reports .NET, you probably need to look a bit further.

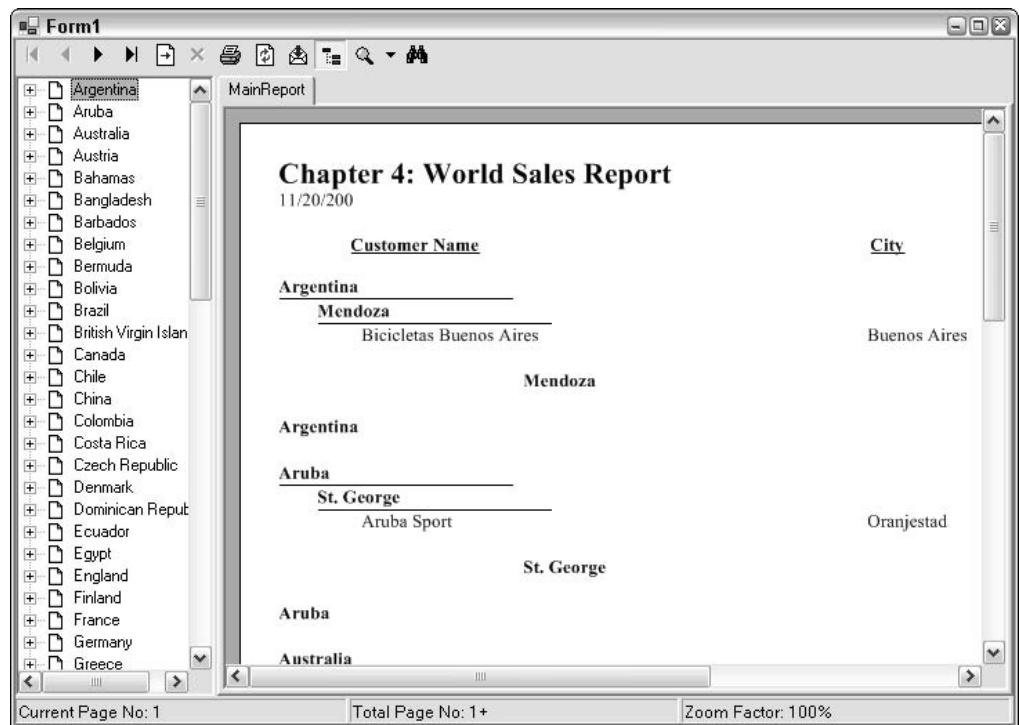


Figure 4-5

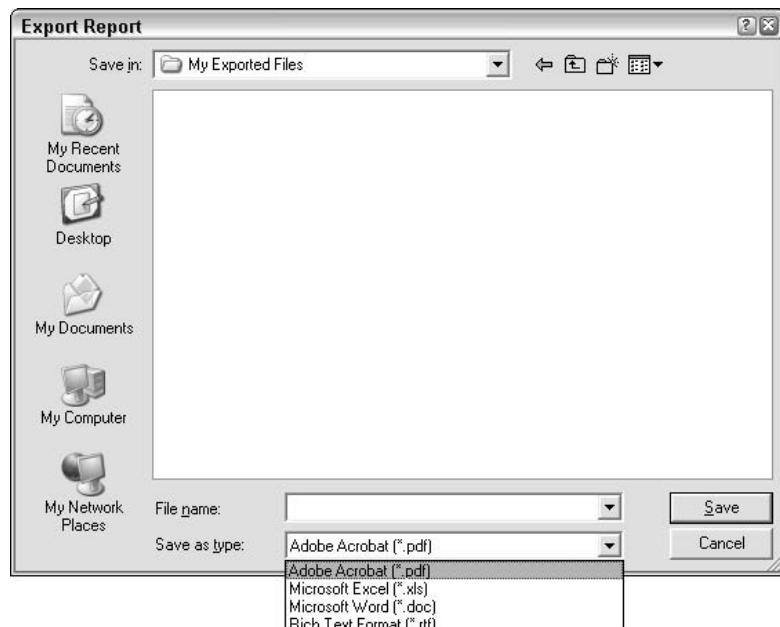


Figure 4-6

Adding a Report to Your Application

Still working in the same solution and project as before, we are going to add a Crystal Report to our application. In the previous walkthrough, we looked at referencing a report that was external to our application through setting the file path and name. Although this is one method of integrating Crystal Reports, actually adding the report file itself to our project makes it easier to edit and integrate reports into our application.

To add our World Sales Report to our project, select Project → Add Existing Item, which will open the shown in Figure 4-7. Change the drop-down list to Show All Files and specify *.rpt for the file name to filter the list to show only the available reports.

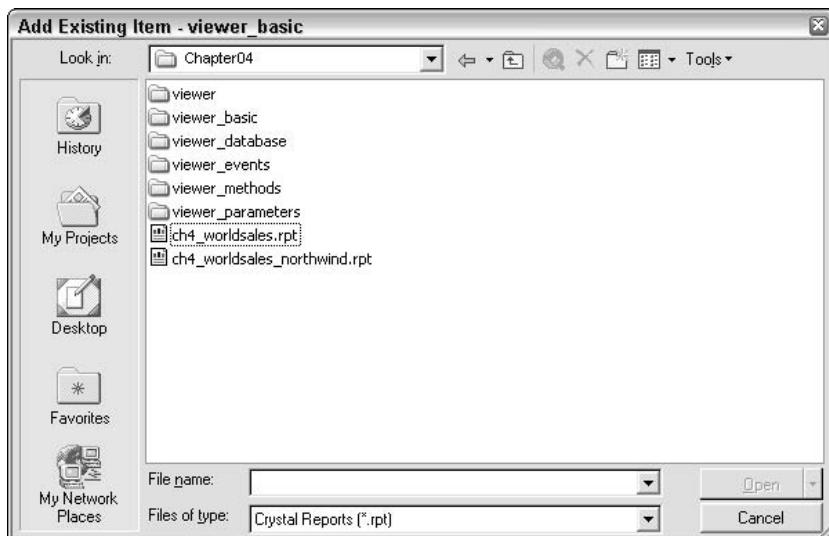


Figure 4-7

Once you have selected the `ch4_worldsales.rpt` report, click Open and this report will be added to your project in the Solution Explorer.

As this report was added to your project, you may have noticed that in addition to the report, another file with the same name as the report (except for a `.vb` extension) is also added, as shown in Figure 4-8.

This file is hidden until you select Show All Files from the Solution Explorer.

This additional file is the report source file and contains a report class specific to this report called `ReportDocument` that is created automatically for you.

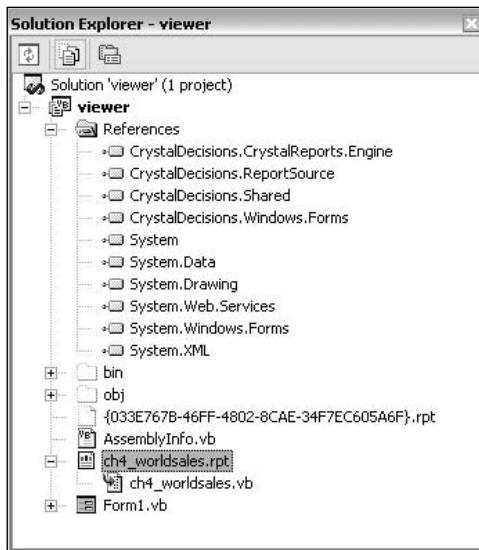


Figure 4-8

Adding the Report Viewer to a Windows Form

Earlier in the chapter, we quickly dragged the Report Viewer onto a form and previewed your first report—the good news is that there is not much more involved in adding the Report Viewer to a form in your application. The Crystal Report Viewer is available from the Windows Forms Toolbox and you can drag it directly onto your form or draw the viewer on your form to the size required.

You can easily add the Crystal Report Viewer to an existing form to display a report side-by-side with other controls, or you could add the report to a new form to have a separate window for previewing reports.

If you are going to use a separate form for previewing reports, try changing the Dock property of the Crystal Report Viewer to Fill so it will fill the entire page. To change this property, right-click the Crystal Report Viewer you have dragged or drawn on your form and select the viewer's properties. From the property page shown subsequently, locate the Dock property and use the drop-down list to select the docking for this component (see Figure 4-9).

You can also add a margin around the docked viewer by setting the Dock Padding properties, which would add blank space around the viewer using the settings and margin you specified.

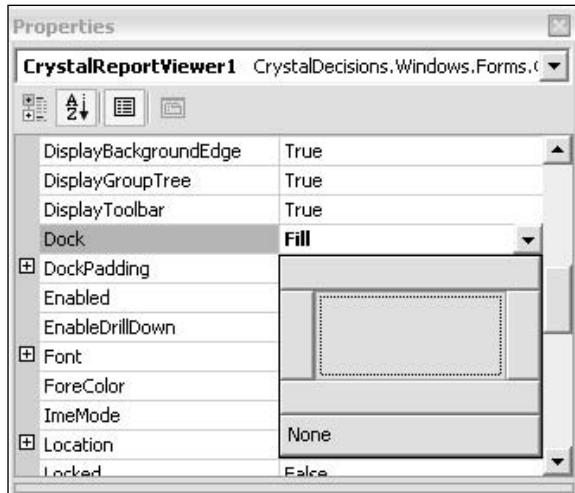


Figure 4-9

Binding a Report to the Report Viewer

With the Report Viewer added to your form, we now need to let the viewer know what report it will be using and this process is called *binding* a report to the viewer. There are four different ways to bind a report to the Crystal Report Viewer:

- By report name
- By report object
- By binding to an untyped report
- By binding to strongly typed report

Binding by Report Name

To bind a report using the report name, all you need to do is set the `ReportSource` property, either through the property page or through the form's code. You will want to add this to the form's `Load` method, which can be accessed by double-clicking the form we are working with. To set the `ReportSource` property, we need to specify the report file name and path, as shown here:

```
CrystalReportViewer1.ReportSource =
"C:\\\\Crystal.NET2003\\\\Chapter04\\\\ch4_worldsales.rpt"
```

Because we have already added the report to our application, we could also bind the report by creating a report object, load the report into the object, and bind the object to the Crystal Report Viewer.

Binding by Report Object

Binding by a report object works slightly differently, depending on whether you have added the report to your project or are referencing it externally. For a report that resides externally to your application, we

Report Integration for Windows-Based Applications

need to first specify an import of the `CrystalDecisions.CrystalReports.Engine`, which will allow us to create the object.

In the Solution Explorer, right-click your project (in our case, `viewer_basic`) and select Properties; click Imports in the left-hand pane.

Under the Common Properties folder, use the Imports option to add the `CrystalDecisions.CrystalReports.Engine` namespace to your project and click Apply and OK to return to your form.

With this namespace imported, we now need to create the report object as a `ReportDocument`, by inserting it into the `CrystalReportViewer1_Load` event, as shown:

```
Private Sub CrystalReportViewer1_Load(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles CrystalReportViewer1.Load  
    Dim myReport = New  
        CrystalDecisions.CrystalReports.Engine.ReportDocument()
```

With our object ready to be used, we can now load the external report file:

```
myReport.Load("C:\\Crystal.NET2003\\Chapter04\\ch4_worldsales.rpt")  
CrystalReportViewer1.ReportSource = myReport  
End Sub
```

If the report you are using has actually been added to your project, a class was automatically generated for the report, so we could use the class instead to bind the report to the viewer (while the import and public declaration remain the same):

```
Private Sub CrystalReportViewer1_Load(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles CrystalReportViewer1.Load  
    Dim myReport As CrystalDecisions.CrystalReports.Engine.ReportDocument  
    myReport = New ch4_worldsales()  
    CrystalReportViewer1.ReportSource = myReport  
End Sub
```

Or if you wanted to eliminate the `myReport` variable, replace that code with the following:

```
CrystalReportViewer1.ReportSource = New ch4_worldsales()
```

Which method you choose will depend on where the report is physically located and how you wish to access it. With any of the preceding examples, you should be able to bind to the report and then run your application—the Report Viewer should display a preview of your report if you are successful.

If you are having trouble binding to a report object, check that the report file name and path are correct, or for a report that has been added to your application, check the report name.

Binding to an Untyped Report

When working with Crystal Reports .NET, you can add individual report files to your project and use and reference them to print reports from your application. Taking this a step further, you could also use these reports as components, which is where we start to look at typing.

As mentioned earlier, when integrating reports into an application, we can either use strongly typed reports or untyped reports. If you have been working with Visual Studio .NET for any length of time, you have probably heard of strongly typed objects. A strongly typed object is predefined with a number of attributes that are specific to that object, giving programmers more structure and a rigorous set of rules to follow, thus making coding easier and the code more consistent.

Within the frame of reference of Crystal Reports .NET, a strongly typed report can be any report that has been added to your project. What determines the type of a report is how it is added and used within your application. When you first add a report to your application, it is considered untyped—the same goes for any external reports that you may reference (such as the report we pointed to earlier, using the file path C:\Crystal.NET2003\Chapter04\ch4_worldsales.rpt).

In the following example, we are going to look at binding to an untyped report. The first step in binding to an untyped report is to add a ReportComponent to the form we are working with.

To add a report component to your form, switch to the Layout view of your form and look in the toolbox under Components. In this section, you should see a component labeled ReportDocument. Drag this component onto your form, which will open the shown in Figure 4-10.

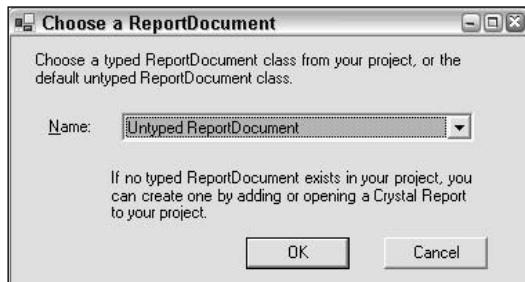


Figure 4-10

This is the we use to set whether our report document component is typed or untyped. If you select Untyped ReportDocument, a report component named ReportDocument1 is created and we can then load a report into this component and bind the component to the viewer.

Again, the code to perform the binding will appear in your form's Form1_Load method and will look something like this:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles MyBase.Load  
    Dim ReportDocument1 As New ReportDocument()  
    ReportDocument1.Load("c:\Crystal.NET2003\Chapter04\ch4_worldsales.rpt")  
    CrystalReportViewer1.ReportSource = ReportDocument1  
End Sub
```

When your application is run and the form previewed, the viewer should show a preview of the report and allow you to print, export, and perform other functions.

If you are experiencing problems binding, check the file name and path of the report you are working with—alternately, if you are binding to a report that exists within your application, check that you have the correct report name.

Binding to a Strongly Typed Report

Finally, if you have added a report to your project and then added a `ReportDocument` component, you can choose to add the report as a strongly typed report component, which probably has the simplest binding method of all. To create a strongly typed report document component, drag the `ReportDocument` component onto your form.

You will then see the same as before, with a drop-down list of all of the available reports that you have added to your project. Select the existing `viewer_basic.ch4_worldsales` report to create a strongly typed report document. From that point, we just need to set the `ReportSource` property in the form's `Load` method:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles MyBase.Load  
    CrystalReportViewer1.ReportSource = ch4_worldsales1  
End Sub
```

Here, `ch4_worldsales1` is the name automatically assigned to the `ReportDocument` component when you added it to your form.

Regardless of which method you choose to bind your report to the viewer, the result is the same. When you start your application and the form is loaded, the Crystal Report Viewer will run the report you have set in the `ReportSource` property and display a preview of the same.

Passing Database Logon Info

Most datasources that Crystal Reports can use are secure, requiring a username, password, and other credentials. When working with the Crystal Report Viewer, we can pass this information through the use of the `TableLogonInfo` collection.

If your report is based on an unsecured database or file, you don't need to pass any logon information at all to your report.

To understand how the Crystal Reports Viewer works with database credentials, consider that each table in your report is its own unique identity, and in turn, its own database credentials. All of these credentials are stored in the `TableLogonInfos` collection, part of the `CrystalDecisions.Shared` namespace, and for each table there is a corresponding `TableLogonInfo` object.

In order to set the database credentials for your report, you will need to loop through all of the tables and set the `ConnectionString` properties within `TableLogonInfo` for each. The properties in the `ConnectionString` class are:

Property	Description
DatabaseName	Gets or sets the name of the database
Password	Gets or sets the password for logging on to the datasource
ServerName	Gets or sets the name of the server or ODBC datasource where the database is located
UserID	Gets or sets a username for logging on to the datasource

Because our Xtreme sample data is held within in an unsecured Access database, we are going to use another report in this example. Included with the sample files for this chapter is a copy of the World Sales Report (ch4_worldsales_northwind) that was created from the Northwind database that ships with SQL Server. You can find this file in the download, through the path Crystal.NET2003\Chapter04\Chapter04\ch4_worldsales_northwind.rpt.

We're going to open a new project and add it to our existing one. Click File → New → Project and select Windows Application. Call the project viewer_database, saving it to Crystal.NET2003\Chapter04. Make sure that the radio button Add to Solution is selected as opposed to Close Solution (see Figure 4-11).

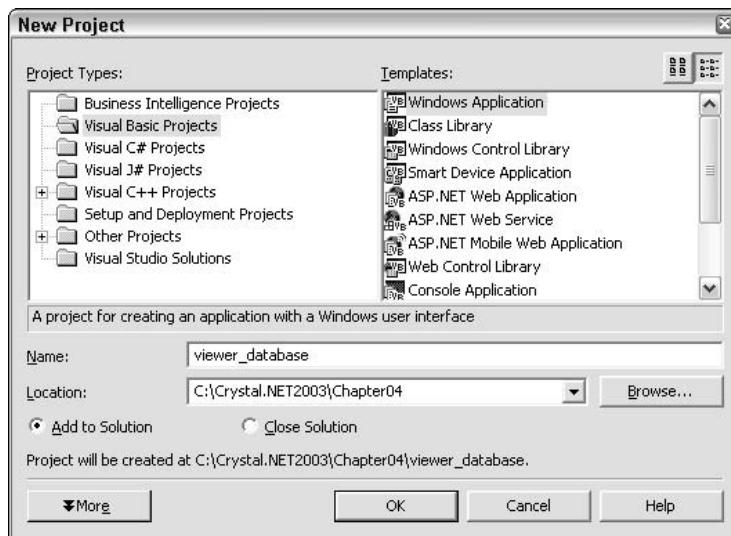


Figure 4-11

Right-click the project name, and click Set As Startup Project. We're now ready to look at database access.

To add ch4_worldsales_northwind.rpt to your project, select Project → Add New Item and browse for the report where you unzipped the downloaded sample files. Add it to the project and click the Open button.

Report Integration for Windows-Based Applications

Now drag and drop a `CrystalReportViewer` onto your form. For the sake of attractiveness, set the `Dock` property to `Fill`.

You will also need to add this report as a component to your form. Switch to the Layout view of your form and look in the toolbox under Components. In this section, you should see a component labeled `ReportDocument`. Drag this component onto your form.

From the drop-down list, select the `ch4_worldsales_Northwind` report and click OK. We are now ready to get on with setting the database credentials for this report.

This report has been written from a single table, `Customer`, for which we are setting the `ConnectionInfo`. The name of our server is `localhost`, off of the Northwind database, with `sa` as the user ID, and no password.

Add this code to the `Form1_Load` method:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
    Dim myReport = New ch4_worldsales_northwind()
    Dim myTableLogonInfos = New CrystalDecisions.Shared.TableLogOnInfos()
    Dim myTableLogonInfo = New CrystalDecisions.Shared.TableLogOnInfo()
    Dim myConnectionInfo = New CrystalDecisions.Shared.ConnectionInfo()

    With myConnectionInfo
        .ServerName = "localhost"
        .DatabaseName = "Northwind"
        .UserID = "sa"
        .Password = ""
    End With

    myTableLogonInfo.ConnectionInfo = myConnectionInfo
    myTableLogonInfo.TableName = "customers"
    myTableLogonInfos.Add(myTableLogonInfo)
    CrystalReportViewer1.LogOnInfo = myTableLogonInfos
    CrystalReportViewer1.ReportSource = myReport
End Sub
```

Make sure that when you are finished setting the `ConnectionInfo` for the table, you specify the table name you are working with prior to using the `Add` method, otherwise you will receive an error message. Also, the username and password for your particular SQL Server may be different.

Compile and run the example. The report should appear, looking identical to the previous examples.

This is a very simple example, as our report only has one table to worry about. If you have a report that features multiple tables or if you don't know the names of the tables, you could also set up a loop to go through each `report.database.table` in `report.database.tables` and set the `ConnectionInfo` properties for each.

In order to get all of the tables in your report and loop through them, you will need to use the Report Engine, which is covered in Chapter 9, "Working with the Crystal Reports Engine."

Setting Report Record Selection

In addition to setting the database credentials for our report, we can also set the record selection formula that is used to filter our report records. The record selection formula within your report is created whenever you use the Select Expert within the Crystal Reports Designer by right-clicking your report and selecting Report → Select Expert, and then clicking the table you wish to see.

This opens up a second , containing the information on that table. You can choose to hide or show the formula on this box, as shown in Figure 4-12.

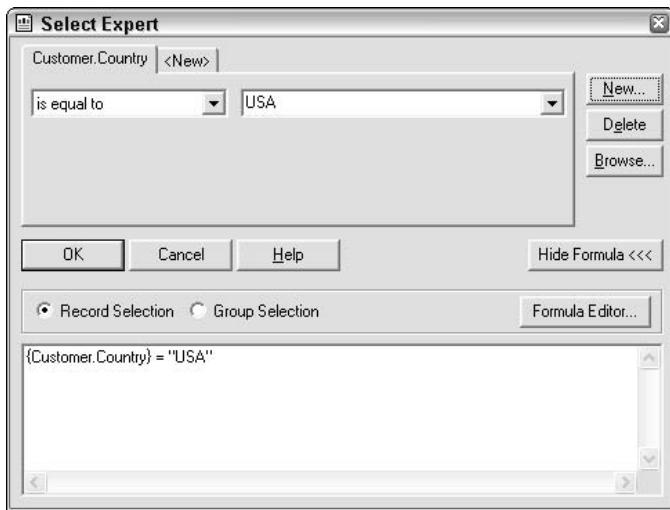


Figure 4-12

Any selections you make in the Select Expert are translated into a record selection formula written in Crystal Syntax, and displayed in the textbox, as you can see in the preceding case.

If you click OK here, then the next time you run the report it will only return customers in the USA.

For more information on creating formulas and Crystal Syntax, turn over to Chapter 8, “Formulas and Logic.”

You can also edit the record selection formula that is generated by right-clicking your report in the Report Designer and selecting Report → Edit Selection Formula → Records.

When your report is run, this record selection formula is translated into the WHERE clause for the SQL statement that is submitted to the database and the results are returned to Crystal Reports. So though you can't change the SQL statement your report is based on, you can control the records that are returned to the report.

Report Integration for Windows-Based Applications

At run time, the `SelectionFormula` property gives us the ability to return or set this value. To return a record selection formula, we would simply request the property as a string, as shown here:

```
CrystalReportViewer1.SelectionFormula.ToString()
```

Using the report and viewer we have been working with, we could set the record selection property when the form loads, before the call to the database, ensuring that only records for customers in the USA are shown:

```
Dim myConnectionInfo = New CrystalDecisions.Shared.ConnectionInfo()  
CrystalReportViewer1.SelectionFormula = "{Customer.Country} = 'USA'"  
With myConnectionInfo  
    ...
```

Alternatively, another way we could use this property is to draw a drop-down list on our form with all of the different countries in the report and allow the user to select which country he or she wants to filter by. When working with record selection, you can come up with some pretty neat ways to deliver one report that could be used for many different types of users.

Working with Parameter Fields

In Chapter 3, “Designing Reports,” we looked at how a Crystal Report can use parameter fields to prompt the user for values that can be displayed on the report, used in record selection criteria, and so on. When viewing a report that has a parameter field present using the `CrystalReportViewer`, the standard parameter will appear when the report is run or refreshed, as shown in Figure 4-13.



Figure 4-13

Although this is the easiest way to implement parameterized reports in your application, the default user interface leaves something to be desired and cannot be customized. Most developers will want to create their own user interface for accepting and passing report parameters. We can set parameter values through functionality found within the standard `CrystalReportViewer` and in the following section, we are going to look at how to programmatically set different types of parameters found in your reports.

To walk through the examples in this section, you will need to have a report that includes parameters and make sure that the parameters are used in the report—either displayed on the report, used in a formula, in the record selection, and so on—because we want to see the results when we actually set these parameter values.

To see this in action, we are going to add yet another new project and add it to our existing one. Click File → New → Project and select Windows Application. Call the project `viewer_parameters`, saving it to `Crystal.NET2003\Chapter4`. Make sure that the radio button Add to Solution is selected as opposed to Close Solution (see Figure 4-14).

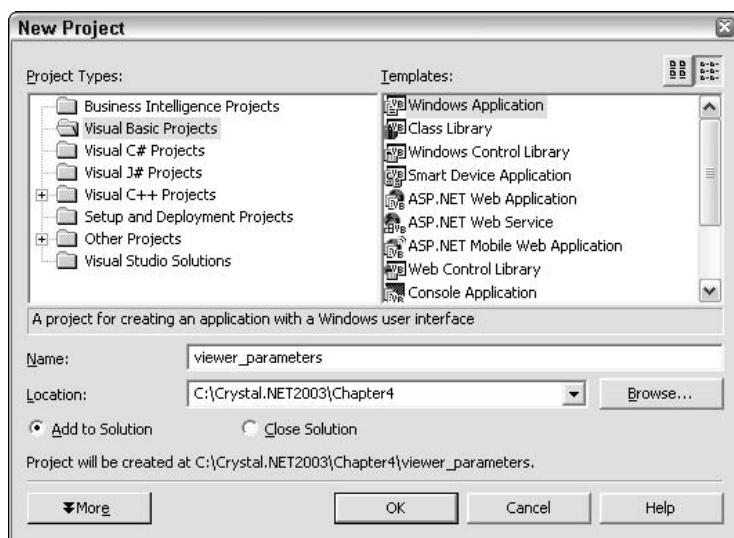


Figure 4-14

Right-click the project name, and click Set As Startup Project. We're now ready to start working with parameters.

The sample report that goes along with this section is called `ch4_parameters.rpt` and to add this report to your project, select Project → Add New Item and browse for the report where you unzipped the downloaded sample files. Add it to the project and click the Open button.

Report Integration for Windows-Based Applications

Now drag and drop a `CrystalReportViewer` onto your form and leave room at the top for some additional fields we are going to add a little later.

You will also need to add this report as a component to your form. Switch to the Layout view of your form and look in the toolbox under Components. In this section, you should see a component labeled `ReportDocument`. Drag this component onto your form.

From the drop-down list, select the `ch4_parameters.rpt` report and click OK. We are now ready to look at how to set the parameters for this report.

To start, think back to our discussion of the different types of parameters you could have in a report—in addition to a parameter’s data type (String, Number, and so on), you can also set options for how the parameter is entered—there are three basic options:

- Discrete* parameters accept single, discrete values—depending on how the parameter field was set up, it can accept only one value or multiple values.
- Ranged* parameters can accept a lower and upper value, selecting everything that is within that range of numbers, letters, and so on.
- Discrete and Ranged* parameters support a combination of the two different types, where you can enter multiple discrete values as well as multiple ranges.

Are you confused now? It can get a bit tricky when you create a report that can accept a combination of the different types, but we are going to walk through these scenarios so you know how to use them with your own reports and parameters.

We are going to start by looking at the easiest implementation—a single, discrete parameter with only one value to enter. Parameters within a report are contained within a collection called `ParameterFields` and within the collection, a set of attributes is stored for each field using the `ParameterField` object. As we walk through this example, we will set properties relating to this object in this collection and then use this collection to set the `ParameterFieldInfo` property of the viewer, as shown here:

```
crystalReportViewer1.ParameterFieldInfo = myParameterFields
```

To set the values for a particular parameter, we need to first dimension both the collection and object. We also need to create a variable to hold the value that we want to pass to the report for this parameter.

Because in this example we are only passing a discrete value to the parameter, we would simply create a variable by dimensioning it as a `ParameterDiscreteValue`. All we need to do at that point is set the value of the parameter equal to some value we have entered or hard-coded and then set the `ParameterFieldInfo` property of the Report Viewer.

Because we want to enter the value ourselves, we would want to put a textbox and a command button on our form to collect and submit the value to be used in our parameter, which would make the form look something like Figure 4-15.

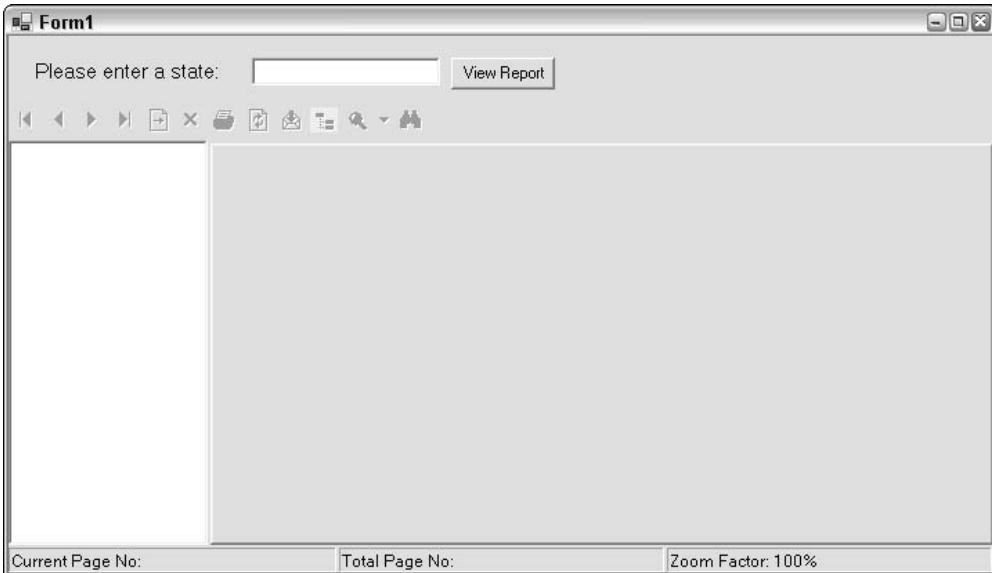


Figure 4-15

To change your form, draw a textbox and a command button on your form, which will be named `TextBox1` and `Command1` by default. Set the properties of the textbox to make it blank when the form appears and change the name of the command button to “View Report,” and then double-click the button on your form to open the code view. This is where we are going to put our completed code that we just walked through. It should look something like this:

```
...
Dim myParameterFields As New ParameterFields()
Dim myParameterField As New ParameterField()
Dim myDiscreteValue As New ParameterDiscreteValue()

myParameterField.ParameterFieldName = "OriginCountry"
myDiscreteValue.Value = TextBox1.Text
myParameterField.CurrentValues.Add(myDiscreteValue)

myParameterFields.Add(myParameterField)

crystalReportViewer1.ParameterFieldInfo = myParameterFields
crystalReportViewer1.refresh

...
```

The next scenario we are going to look at is for multiple discrete variables; in this situation, we would use the same methods, only instead of setting a single value using the `ParameterDiscreteValue` variable, we would continually redeclare this variable and use it to push values into the `ParameterField` object, as shown here:

Report Integration for Windows-Based Applications

```
...  
  
Dim myParameterFields As New ParameterFields()  
Dim myParameterField As New ParameterField()  
Dim myDiscreteValue As New ParameterDiscreteValue()  
  
myParameterField.ParameterFieldName = "OriginCountry"  
  
myDiscreteValue.Value = "Australia"  
myParameterField.CurrentValues.Add(myDiscreteValue)  
  
myDiscreteValue= New ParameterDiscreteValue()  
  
myDiscreteValue.Value = "New Zealand"  
myParameterField.CurrentValues.Add(myDiscreteValue)  
  
myParameterFields.Add(myParameterField)  
  
crystalReportViewer1.ParameterFieldInfo = myParameterFields  
...
```

In this example, we have used the variable twice to push both “Australia” and “New Zealand” to the report. Remember, in your actual report design, you must specify whether the parameter option is a discrete, multi-value discrete, ranged, and so on. Otherwise you will receive an error message when running your report.

Finally, our last type of parameter is a ranged parameter, which can be set using the same method, except instead of a discrete value, we are going to pass in a range, with a start and end value, as shown here:

```
...  
  
Dim myParameterFields As New ParameterFields()  
Dim myParameterField As New ParameterField()  
Dim myDiscreteValue As New ParameterDiscreteValue()  
Dim myRangeValues as New ParamterRangeValues()  
  
myParameterField = New ParameterField()  
myParameterField.ParameterFieldName = "LastYearsSales"  
  
myRangeValues.StartValue = 100000  
myRangeValues.EndValue = 500000  
  
myParameterField.CurrentValues.Add(myRangeValue)  
myParameterFields.Add(myParameterField)  
  
...
```

You can also combine the two methods depending on the type and number of parameters you have created in your report. You could have two parameters, for example, that accepted discrete values and one that accepted a range and still yet another that accepted a combination of the two. The methods used to set the values for these parameters are the same, so users can easily enter parameter values through their own application and filter the report content.

Speaking of different types of users, the next section deals with how to customize the appearance and layout of the viewer itself. So in addition to showing them only the records they want to see, you could also give them their own custom viewer with which to view the resulting report.

Customizing the Appearance and Behavior of the Report Viewer

The `CrystalReportViewer` class contains all of the properties, methods, and events that relate to the viewer itself, its appearance, the methods that are used to make the viewer perform certain actions (like refresh or print a report), and events that can be used to determine when a particular event (such as drill-down or refresh) has occurred. To start learning how to work with the viewer, we are going to start with the basic properties and move on from there.

When previewing your report, you may notice that there is a standard set of icons and layout that appears on your viewer by default, but you can control most of the aspects of the viewer by setting a few simple properties for the Crystal Report Viewer in the Properties window, as shown in Figure 4-16.

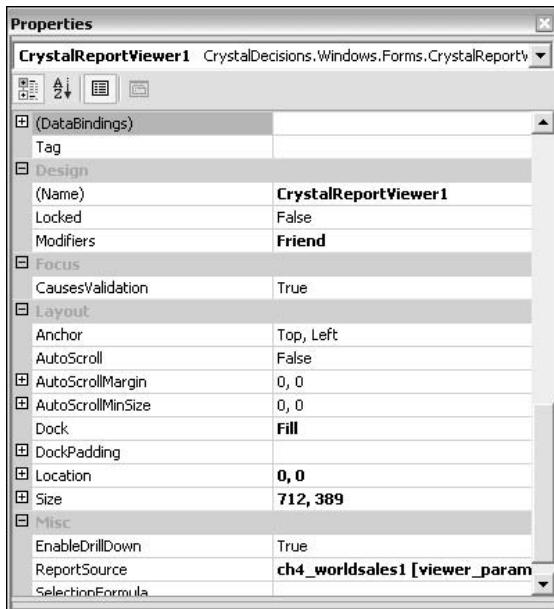


Figure 4-16

The area at the top of the viewer is the toolbar, which can be shown or hidden as an entire object, or you can choose to only show certain icons. On the left-hand side is a Group Tree, generated by the grouping that you have inserted into your report. The properties that control these general properties are Boolean and are listed below:

Report Integration for Windows-Based Applications

Property	Description
DisplayBackgroundEdge	For showing the off-set edge around your report when previewing
DisplayGroupTree	For showing the group tree on the left side of the viewer
DisplayToolbar	For showing the entire toolbar at the top of the viewer

All of these properties default to `True` and you cannot change the position of any of these elements; they are fixed in place on the viewer. You can, however, hide the default toolbar and create your own buttons for printing, page navigation, and other functions, and we'll look at that a little later in the chapter.

For the icons within the toolbar, you can also set simple Boolean properties to show or hide a particular icon, as shown here:

- `ShowCloseButton`
- `ShowExportButton`
- `ShowGotoPageButton`
- `ShowGroupTreeButton`
- `ShowPageNavigateButtons`
- `ShowRefreshButton`
- `ShowTextSearchButton`
- `ShowZoomButton`
- `ShowPrintButton`

So a typical use of these properties is where you want to give users a view-only preview, with no printing or exporting options and no option to refresh the report. Going back to our original report (`ch4_worldsales`), we could easily set a few properties before you set your `ReportSource` property to make this happen.

Double-click anywhere on your form to open the code view and in the form's `Load` method, enter the following:

```
...
End With
myTableLogonInfo.ConnectionInfo = myConnectionInfo
myTableLogonInfo.TableName = "customers"
myTableLogonInfos.Add(myTableLogonInfo)
CrystalReportViewer1.LogOnInfo = myTableLogonInfos
CrystalReportViewer1.ReportSource = myReport
CrystalReportViewer1.DisplayGroupTree = False
CrystalReportViewer1.ShowExportButton = False
CrystalReportViewer1.ShowRefreshButton = False
CrystalReportViewer1.ShowPrintButton = False
CrystalReportViewer1.ReportSource = New ch4_worldsales_northwind()

End Sub
```

Chapter 4

When the report is previewed, it will appear as shown in Figure 4-17.

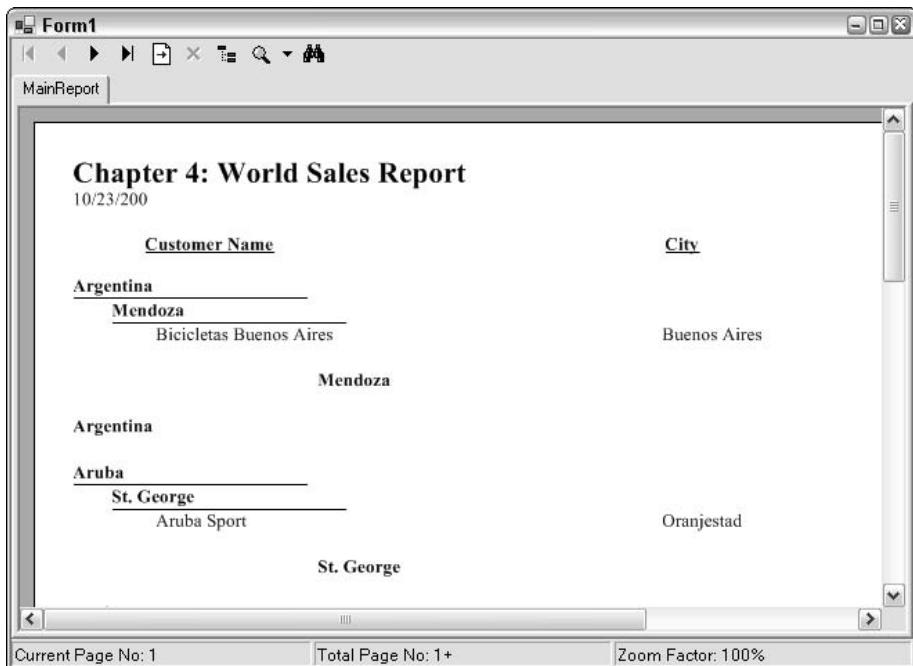


Figure 4-17

Keep in mind that you can set these properties any time prior to the report preview. You could store individual user or security settings in your application data and then set the appropriate properties prior to viewing the report. This is just one example of where we can customize how a report is presented to the user; the next section on the methods available within the viewer takes that discussion one step further.

Viewer Methods

When working with the Crystal Report Viewer, we have a number of methods available to us, which will allow us to integrate specific viewer functions into our application. As we move through this section, keep in mind that these methods can be used to create your own look and feel for the report preview window, as shown in Figure 4-18.

During the course of this section, we will actually be looking at the code behind the custom viewer shown here, so it is probably not a bad idea to start a new project within this chapter's solution file. To create a new project from within Visual Studio, select File → New → Project and from Visual Basic Projects, select Windows Application and specify a name, as shown in Figure 4-19, (in the sample code, we have called this project `viewer_methods`) and location for your project files. Remember to set this project as your startup project.

Report Integration for Windows-Based Applications

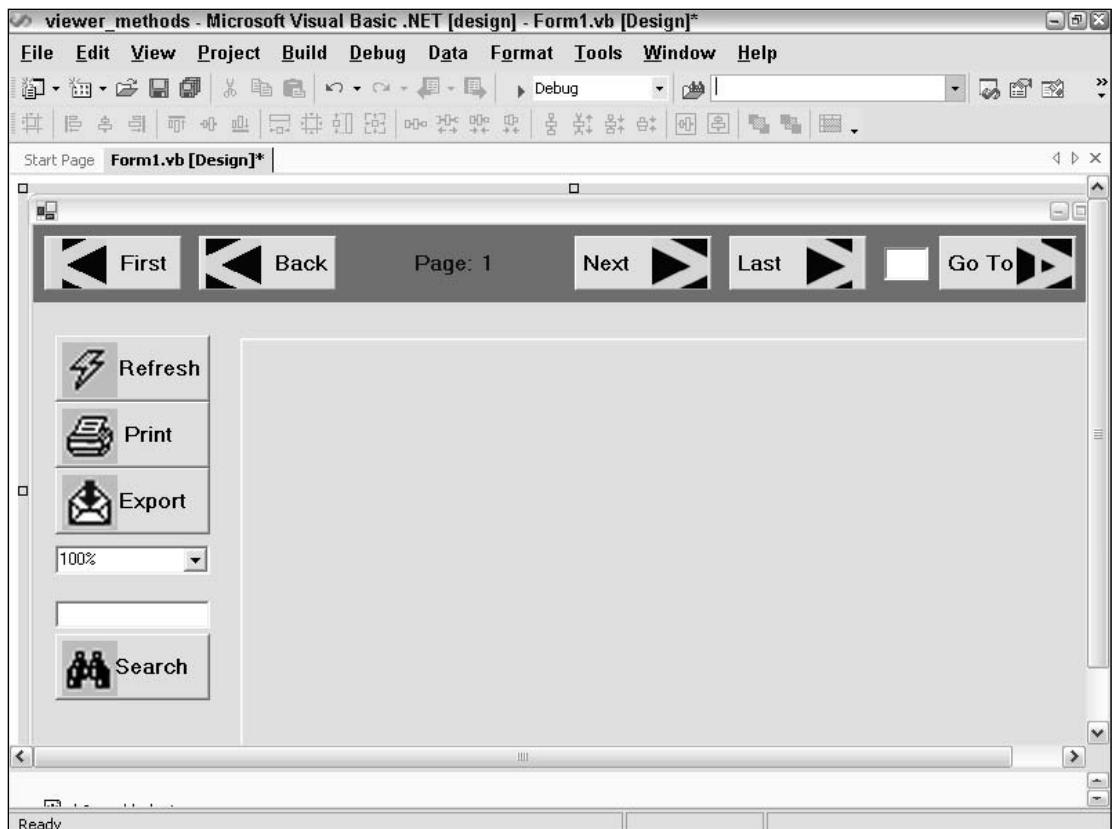


Figure 4-18

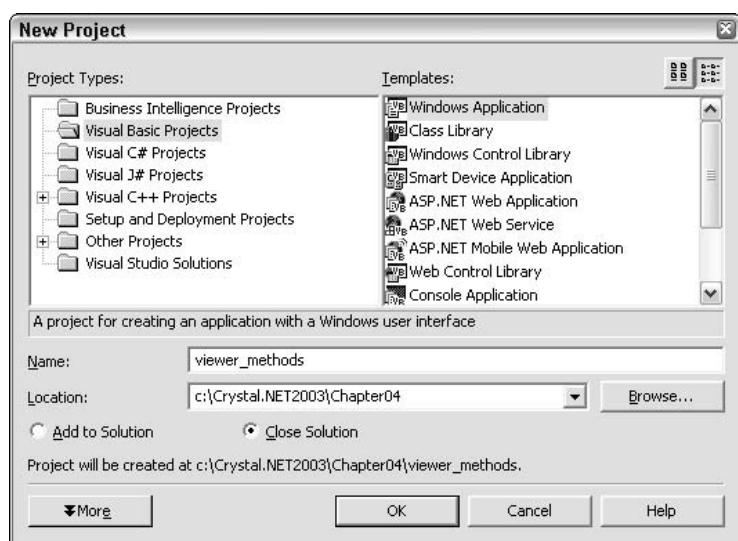


Figure 4-19

Chapter 4

Once your sample project has been created, add the Crystal Report Viewer to the default form that is created and copy or add the ch4_worldsales.rpt to your project. Set the ReportSource property to point to this report, using the property page shown in Figure 4-20. We are now ready to get started.

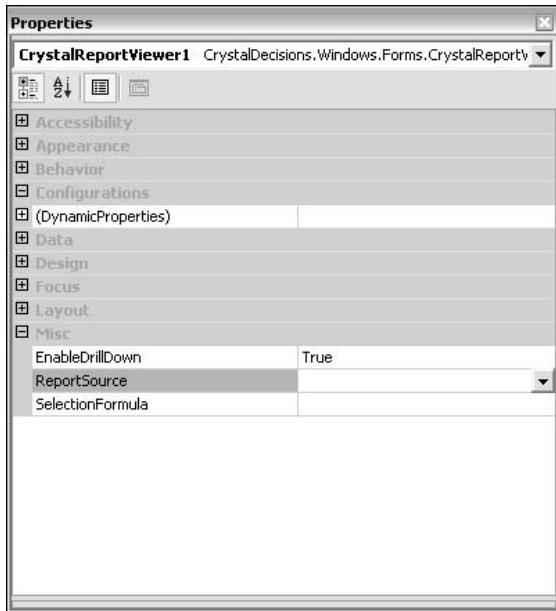


Figure 4-20

The first thing we need to do to emulate the custom viewer shown earlier is to set the `DisplayToolbar` and `DisplayGroupTree` properties to `False`.

Next, we need to add additional buttons for some of the functions normally associated with the buttons shown on the viewer toolbar—the buttons for the next page, previous page, print, export, and so on—using the custom viewer shown in Figure 4-21 as a guide. We will walk through each of these later in this chapter.

The tangible benefit of using the methods described subsequently and your own form design is that you have more flexibility in how the report appears when viewed and you can match the viewer's user interface to your own application.

Printing a Report

To print a report, there is a simple `PrintReport` method that will invoke a standard Windows printer to select where you would like to print your report, how many copies, and other functions.

To add this code to your custom viewer, drag and drop a button onto your form, and name it `Print_Button`. Change the `Text` property to `Print`. Double-click the Print button you have dropped onto your form and enter the following code in its `Click` event:

```
CrystalReportViewer1.PrintReport
```

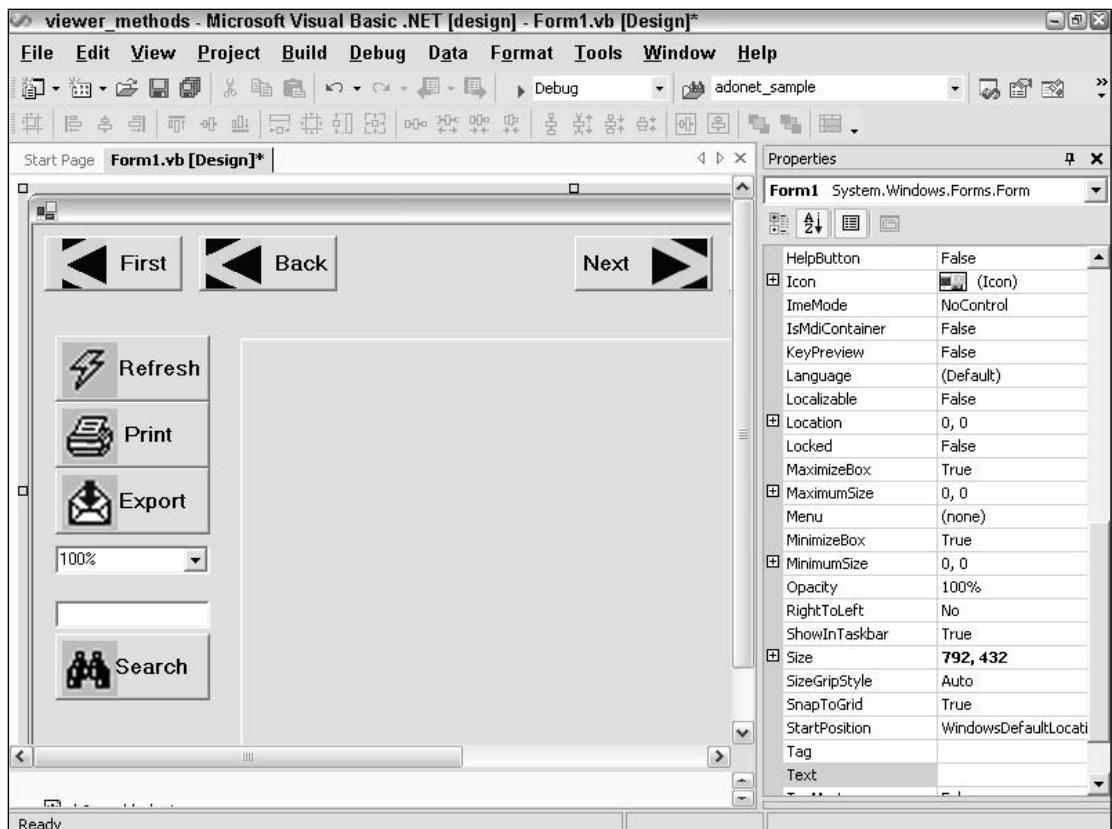


Figure 4-21

This will open a standard Windows print that will allow you to print your report. If you need to access advanced print options (like printing two pages to separate paper trays) or if you want to control the print process manually, you will need to use the Report Engine to do so, which is covered in Chapter 8, "Formulas and Logic."

Refreshing the Data in a Report

When a report is refreshed, it goes back to the database for the most current set of data available and runs again. Drag and drop a button onto the form, and call it `Refresh_Button`. Change the text to Refresh. To refresh from the Crystal Report Viewer, you can add `RefreshReport` method to the Refresh button you have created on your custom viewer form:

```
CrystalReportViewer1.RefreshReport
```

If your report takes a while to run or if you are concerned about database traffic and load, you may want to consider removing this as an option from your viewer, or even changing the properties of the standard viewer so that the Refresh icon does not appear at all, using the syntax `CrystalReportViewer1.ShowRefreshButton = False.`

Exporting a Report

Crystal Reports .NET features a rich export functionality, which is partially exposed in the Crystal Report Viewer. From the viewer, we can call the `ExportReport` method to open a Save as... and export your report into one of four formats:

- Adobe Acrobat (PDF)
- Microsoft Excel (XLS)
- Microsoft Word (DOC)
- Rich Text Format (RTF)

In compatibility testing, the export formats for Microsoft Word and Excel work well with Office 97+, and Rich Text Format can be used by just about any word-processing application (including Word, WordPad, WordPerfect, and so on). For Adobe Acrobat, a version 3.0 or above reader is recommended and the output is consistent across version 3.0–6.0.

So, let's put this functionality into our custom viewer. Drag and drop a button onto the form once more, this time calling it `Export_Button` and setting the text to `Export`. Once again, click the button to open its code event. Insert the following:

```
CrystalReportViewer1.ExportReport()
```

When the `ExportReport` method is used, the shown in Figure 4-22 will appear and allow you to select an export format from a drop-down list and select where the file is to be saved.

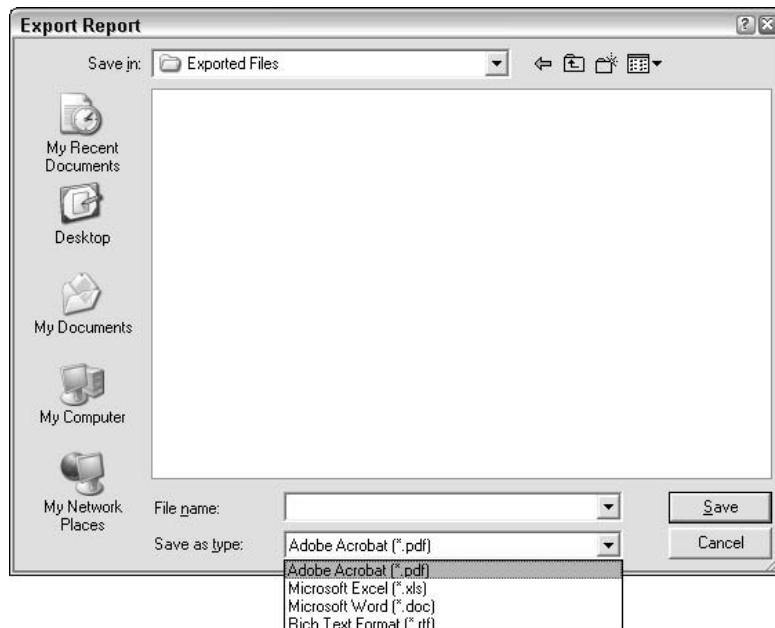


Figure 4-22

Report Integration for Windows-Based Applications

Once the file has been saved, a message box will appear, advising you that the export is complete. You can then use the associated application to open the exported file.

Page Navigation and Zoom

To start, you probably will want to know what page you're are on at some point. Luckily for us, the Crystal Report Viewer has a simple method called `GetCurrentPageNumber` that allows us to get the page number of the page we are currently viewing.

In the custom viewer we are working with, we are going to place a label on the form (we'll call it `PageNo_Label`) that contains the page number. Initially, we'll set this to `Page: 1` using the `Text` property, but after that, this can be set dynamically using:

```
PageNo_Label.Text = "Page: " &
    CrystalReportViewer1.GetCurrentPageNumber.ToString
```

This method should be called after moving through the report pages, so it should be placed after the code for each of the following buttons (and the Refresh button, of course—do this now).

So, we'll create the following four buttons, and place them on our form.

Button Name	Button Text Property Value
FirstPage_Button	First Page
BackPage_Button	Back
NextPage_Button	Forward
LastPage_Button	Last Page

In order to navigate through the pages of our report, we have a number of methods that can be called without any additional parameters, as shown subsequently:

- `ShowFirstPage`
- `ShowLastPage`
- `ShowNextPage`
- `ShowPreviousPage`

So to put code behind our navigation buttons on our custom form (in this case, the Forward button), we could use the `ShowNextPage` method.

```
CrystalReportViewer1.ShowNextPage()
PageNo_Label.Text = "Page: " &
    CrystalReportViewer1.GetCurrentPageNumber.ToString
```

Compile and run this. You should be on page two of the report, and the label should inform you of this. Now populate the remaining buttons with the code, remembering to set the correct method for each button.

Chapter 4

These methods do not return a result, so to determine what page you are currently on, we would have to use the `GetCurrentPageNumber` method immediately after calling the first method, which will return the page you are currently viewing. Unfortunately, we don't have a way to get the total page count, unless you were to use `ShowLastPage` to go to the last page, use the `GetCurrentPageNumber` method, and then store the number of the last page in a variable somewhere in your code, but that is a lot of work for one little number.

For navigating to a specific page, `ShowNthPage` allows us to pass a specific page number to the method, as shown here, emulating the functionality of the `ShowNextPage` method.

```
Dim currentPage
currentPage = CrystalReportViewer1.GetCurrentPageNumber
CrystalReportViewer1.ShowNthPage(currentPage + 1)
' This will take you to the next page
```

In the custom viewer we are working with, draw a text box onto the form, naming it `PageNo_TextBox`. The point of this text box is to allow the user to enter a page number and then click the Go To button to go to a specific page. Drag and drop a button on the form next to the text box, naming the button `GoTo_Button` and labelling it Go To.

Assuming that the textbox you have drawn on your form is called `PageNo_TextBox`, the following code, placed behind the Go To button, checks to see if a page number has been entered. If something has been entered, the `ShowNthPage` method is then called to jump to a specific page.

```
If PageNo_TextBox.Text <> "" Then
    CrystalReportViewer1.ShowNthPage(PageNo_TextBox.Text)
    PageNo_Label.Text = "Page: " &
        CrystalReportViewer1.GetCurrentPageNumber.ToString
    PageNo_TextBox.Text = " "
Else
    MsgBox("Please enter a page number to jump to",
        MsgBoxStyle.Exclamation, "Please enter a page number")
    PageNo_TextBox.Text = " "
End If
```

Compile and run, and you should see that this functionality is now implemented.

In addition to page navigation, you also have the ability to choose the zoom factor that is applied to your report. By default, the zoom is set to 100% of the report size unless you specify otherwise. In the following example, we will add a combo box to the form to allow the user to select a particular zoom factor for viewing.

The name of the combo box is `ComboBox_Zoom`. Assign the `Text` property with the value 100%, and click the `Items` property. The String Collection Editor should now open. Enter the following strings, one per line:

- 25%
- 50%
- Full Size
- 200%

Report Integration for Windows-Based Applications

Now, we move on to the business of selecting and setting a zoom factor based on the index of the item that has been selected. Double-click the combo box and enter the following code:

```
With CrystalReportViewer1
    Select Case ComboBox_Zoom.SelectedIndex
        Case 0
            .Zoom(25)
        Case 1
            .Zoom(50)
        Case 2
            .Zoom(100)
        Case 3
            .Zoom(200)
    End Select
End With
```

You also may want to consider adding the option to let users select their own zoom factor. Keep in mind that 50% is about the lowest resolution at which a report can be read legibly with a 12-point font used in the report itself—if you are concerned about how the report will appear when viewed, you may also set the minimum zoom required to view the report as it should appear.

Searching Within a Report

Another powerful navigation feature can be found in the `SearchForText` method within Crystal Reports .NET, which will allow you to search for a specific string that appears in your report. In our custom viewer, we will add a textbox (`SearchString_TextBox`) for the user to enter a search string, as well as a Search button (`Search_Button`) to kick off this method. Drag both of these items onto the form, and set their properties appropriately.

The code behind the search button looks like this:

```
If SearchString_TextBox.Text <> "" Then
    CrystalReportViewer1.SearchForText(SearchString_TextBox.Text)
    pageNo_Label.Text = "Page: " &
        CrystalReportViewer1.GetCurrentPageNumber.ToString
    SearchString_TextBox.Text = " "
Else
    MsgBox("Please enter a search string to search for",
        MsgBoxStyle.Exclamation, "Please enter a string to search for...")
    SearchString_TextBox.Text = " "
End If
```

We first check to see if a value is entered, and if so, the `SearchForText` method is called, passing the search string that was entered. The Crystal Report Viewer will search the entire report and when the value is found, go directly to the page on which it appears and highlight the value. This method can be called repeatedly to find all of the occurrences of a particular string—each time it finds the string in your report, it will jump to that page and highlight where the value appears.

Using our World Sales Report and searching on Hong Kong should jump to the first company with a region of Hong Kong and highlight the value, as shown in Figure 4-23.

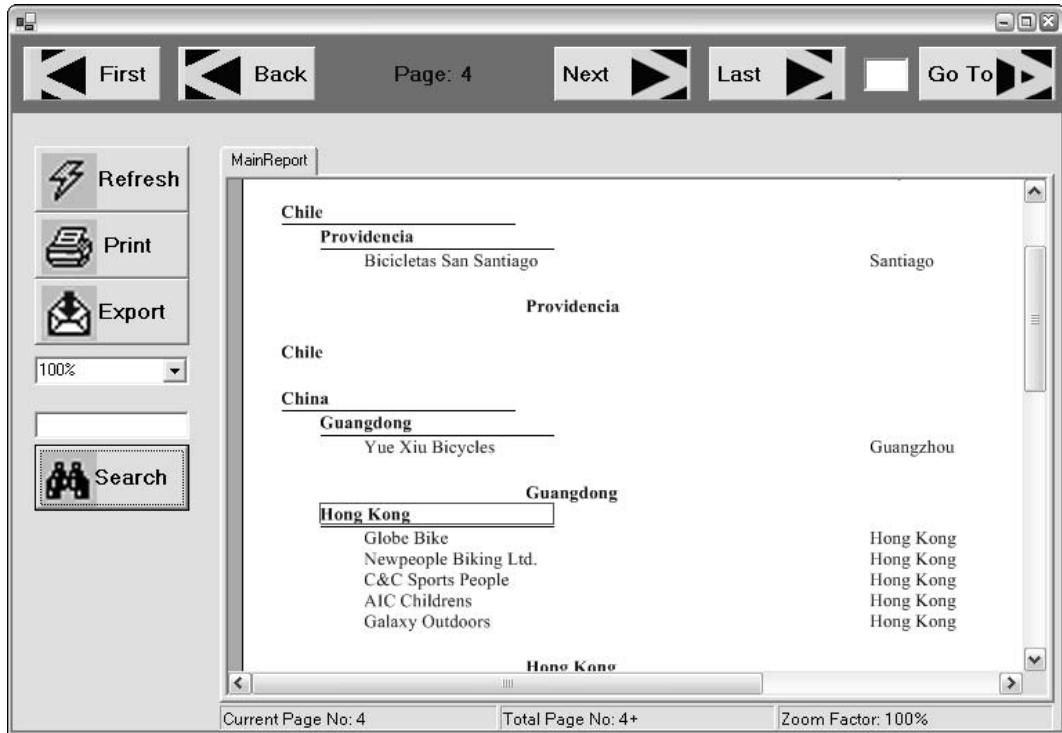


Figure 4-23

Using Viewer Events

Viewer events provide the ability to track the firing of different events—for instance, when users navigate through the pages of the report, or when they refresh the report. These events can then be used to fire other code from within your application.

Although all of the different events have their own unique properties and methods, they all inherit a common property called `Handled`. This is a Boolean value that is used to determine whether the event was fired and subsequently handled.

In the following section, we will be looking at all of the available events associated with the viewer and their common uses. Again, because this is a new set of functionality contained within the viewer, we are going to create another project to hold all of the code and forms related to this section.

To create a new project from within Visual Studio, select File → New → Project and from Visual Basic Projects, select Windows Application and specify a name (in the sample code, we have called this project `viewer_events`) for your project files. Set this as your startup project within the solution.

Once your sample project has been created, add the Crystal Report Viewer to the default form that is created and copy or add the `ch4_worldsales.rpt` to your project. Set the `ReportSource` property to point to this report and let's get coding.

Report Integration for Windows-Based Applications

For all of these events, we are going to place the code behind our form and when a particular event is fired, the code will be run.

Page Navigation Events

For page navigation, the `NavigateEventArgs` class provides the properties we need to work with the `Navigate` event, including:

Property	Description
<code>CurrentPageNumber</code>	Returns the current page number
<code>NewPageNumber</code>	Gets or sets the new page number

In the following example, the `Navigate` event would fire when a user changed the page within the viewer, resulting in a message box that would show the current page, and the page being navigated to.

Insert the following subroutine into your form:

```
Private Sub CrystalReportViewer1_Navigate(ByVal source As Object, ByVal
    MyEvent As CrystalDecisions.Windows.Forms.NavigateEventArgs) Handles
    CrystalReportViewer1.Navigate

    If MyEvent.NewPageNumber <> 1 Then
        MsgBox ("Current page: " & MyEvent.CurrentPageNumber & " New Page: " &
            MyEvent.NewPageNumber)
    End If
End Sub
```

Compile and run with this code. When the form opens with the report, click the last page icon. You should see a message box similar to the one in Figure 4-24.

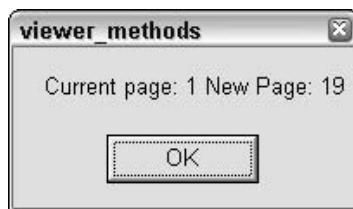


Figure 4-24

This event could be used to determine when the first page was viewed, and pop up another form with an explanation of the report and its contents, or used to perform a task in the background (like logging page views) while the user is viewing the report.

Chapter 4

Refresh Events

The ReportRefresh event has no arguments other than the inherited Handled property. It can be used to build metrics on how often a report is run or refreshed, and to pass information to users about the report before they launch a refresh, as shown here:

```
Private Sub CrystalReportViewer1_ReportRefresh(ByVal source As Object, ByVal  
    MyEvent As CrystalDecisions.Windows.Forms.ViewerEventArgs) Handles  
    CrystalReportViewer1.ReportRefresh  
    MsgBox ("Please be advised this report takes up to 2 minutes to run.")  
End Sub
```

Refresh events are also key to improving application and data performance; if your database is only updated once a day (or once a month), you can keep track of how many times a user attempts to hit the database, and simply remind users with an information box that the data will remain the same during the day, regardless of how many times they hit the refresh button!

Search Events

When a user searches for a report value, either through the standard icon on the toolbar or through your own method call, the Search event is fired. The arguments for the Search event are:

Property	Description
Direction	Gets or sets the direction in which to search. This can be either Backward or Forward.
PageNumberToBeginSearch	Gets or sets the page number on which to start searching.
TextToSearch	Gets or sets the text to search for in the report.

So by using these event arguments, you could keep a record of what values users searched for. An example of getting the text that is being used in the search is included in the following:

```
Private Sub CrystalReportViewer1_Search(ByVal source As Object, ByVal  
    MyEvent As CrystalDecisions.Windows.Forms.SearchEventArgs) Handles  
    CrystalReportViewer1.Search  
    MsgBox ("You searched for " & MyEvent.TextToSearch )  
End Sub
```

Viewer Events

The Load event is fired whenever the Report Viewer is initialized from a Windows Form and has no other arguments except for Handled. You can use this event to fire other sections of code or launch additional windows for help, or a description of the report, as shown here:

```
Private Sub CrystalReportViewer1_Load(ByVal sender As System.Object, ByVal
    MyEvent As System.EventArgs) Handles CrystalReportViewer1.Load
    MsgBox ("This report shows monthly sales broken down by region")
End Sub
```

Again, you could also use this event for logging as well.

Zoom Events

For those times that the user changes the zoom factor for a particular report, the `ViewZoom` event fires and has only one argument in `ZoomEventArgs`. The `NewZoomFactor` property will get or set the magnification factor for the viewer, as shown here:

```
Private Sub CrystalReportViewer1_ViewZoom(ByVal source As Object, ByVal
    MyEvent As CrystalDecisions.Windows.Forms.ZoomEventArgs) Handles
    CrystalReportViewer1.ViewZoom

    Select Case MyEvent.NewZoomFactor
        Case "25"
            MsgBox ("You have selected 25%")
        Case "50"
            MsgBox ("You have selected 50%")
        Case "100"
            MsgBox ("You have selected full size")
    End Select
End Sub
```

Drilling into Report Details

If you are working with a report that has groups inserted, you can drill down within the viewer to show the detailed records that make up that group. By default, these details are visible anyway, as shown in Figure 4-25.

When you drill down into a group, a separate tab is opened within the preview window, showing only the group you have selected. For summary reports, you may want to hide the details and allow users to drill down if they need more information.

This provides an easy way to cut down on report development; instead of multiple reports for different regions, for example, you could create one report and then let the users drill into it and print the section they wanted to see. In the following example, the user has drilled down into Australia, which opens another tab in the viewer, allowing the user to see the regions within Australia.

When you double-click a group or summary and drill down into a report, the `Drill` event is fired and can be used to return the name of the group, the level, or other information. There are a number of properties associated with `DrillEventArgs`, including:

Chapter 4

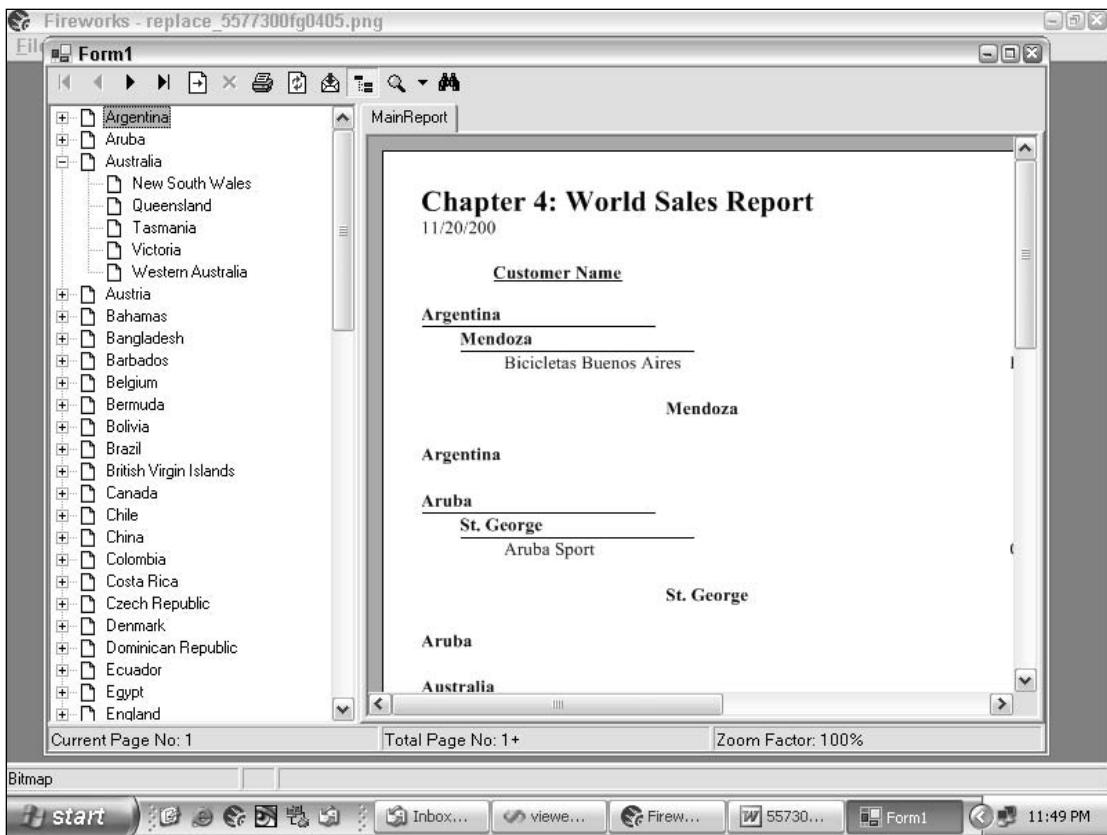


Figure 4-25

Property	Description
CurrentGroupLevel	Returns the group level that was drilled into
CurrentGroupName	Returns the name of the group that was drilled into
CurrentGroupPath	Returns the group number and group level that was drilled into
NewGroupLevel	Returns the target group level that is being drilled into
NewGroupName	Returns the target group name that is being drilled into
NewGroupPath	Returns the target group number and group level that is being drilled into

CurrentGroupNamePath and **NewGroupNamePath** are included within **DrillEventArgs**, but are reserved for future use.

Report Integration for Windows-Based Applications

To see the Drill event in action, you will need to have a report that has at least one group inserted and a section where the details are hidden (not suppressed). In addition, in the Crystal Report Viewer, the EnableDrillDown property must be True. The Drill event will fire whenever you drill down into one of the groups on your report and can be used to determine the group name and what level has been drilled into, among other things.

Drill events can be used to launch other forms or processes; for example, when a user drills down on a Country group, you could display a form giving a background to the country, its currency, and other pertinent information.

The following code demonstrates the Drill event being used to display an information box, containing information on where the user was drilling from and the target they were drilling to.

```
Private Sub CrystalReportViewer1_Drill(ByVal source As Object, ByVal MyEvent  
    As CrystalDecisions.Windows.Forms.DrillEventArgs) Handles  
    CrystalReportViewer1.Drill  
  
    MsgBox("You drilled into " & MyEvent.NewGroupName() & (Chr(13) & Chr(10)),  
        MsgBoxStyle.Information, "Drill Down Event")  
  
End Sub
```

Drilling Down on Subreports

Multiple subreports can be inserted into a main report and provide a way of combining disparate information on a single report. A subreport within a Crystal Report is actually a report in its own right, with its own page numbering, sections, and other information.

There are a number of subreport events that can be used as users drill through a report with subreports, including:

Property	Description
CurrentSubreportName	Returns the name of the subreport that was drilled into
CurrentSubreportPageNumber	Returns the page number that the subreport is on
CurrentSubreportPosition	Returns the location in the viewer where the subreport is
NewSubreportName	Returns the name of the subreport that is being drilled into
NewSubreportPageNumber	Returns the page number to drill into the subreport
NewSubreportPosition	Returns the location in the viewer where the subreport is to drill into

Chapter 4

Using the properties in the previous table, you could determine the name of a report that had been drilled into and then use the same for logging and launching other forms. Our report does not contain a subreport, but the methods remain the same.

For changing elements of a subreport, we would need to use functionality from the Crystal Report Engine, covered in Chapter 9, "Working with the Crystal Reports Engine."

Dealing with Report Exceptions

The `HandleException` event fires whenever you are viewing a report and the viewer encounters any errors or exceptions. This could be caused by a datasource not being available, the report file itself being moved to a different location (if external to your application), or any other error that may occur.

There are a number of arguments that are associated with this event, including

Property	Description
<code>Exception</code>	Returns the exception data for the exception that has occurred
<code>UserData</code>	Returns or sets any type of data that can be used to override what is done in the handling of an exception

The `UserData` property is a generic object that can be used to override the error handling that the viewer would normally do. For example, if you were using an Access database and it had been moved and was no longer available, you could set the `UserData` property to a string containing the location to the `UserData`, and that particular database location would be used.

So, to trap these and other types of errors, you can set up an error handler event and then use the exception to return the error message:

```
Public Sub CrystalReportViewer1_HandleException(ByVal source As Object,  
    ByVal MyEvent As CrystalDecisions.Windows.Forms.ExceptionEventArgs)  
Handles CrystalReportViewer1.HandleException  
  
    Dim err As Exception  
    err = myEvent.Exception  
    MsgBox("An error has occurred with your report:" & (Chr(13) & Chr(10)) &  
        err.ToString, MsgBoxStyle.Critical, "Exception Event")  
End Sub
```

You also may want to consider tying context-sensitive help (where the help topic directly relates to the error message produced) to the error as well, to give the user a more complete description of what the error really means.

Summary

In this chapter, you have had a look at integrating reports into Windows applications, starting with basic integration with the Crystal Report Viewer for Windows forms. In terms of ease of use and functionality, the Crystal Report Viewer provides most of the functionality you will need for view-only reporting implementations. In addition to the standard viewer functionality, we also looked at how you could use the properties, methods, and events of the viewer to customize the look and feel or even create your own custom viewer that matches your own application's user interface.

So what is next? For more advanced integration topics and greater control over the report itself, you may want to consider flipping over to Chapter 8, "Formulas and Logic," to start learning how the Report Engine can be used to control your report's contents and appearance.

If you also develop Web applications, you are probably keen to get into the next chapter, but keep in mind that some of the same concepts we just covered in this chapter (and in Chapter 9, "Working with the Crystal Reports Engine" with the Report Engine topics) will also apply to Web applications, which we will be looking at next.

5

Report Integration for Web-Based Applications

ASP .NET provides a flexible programming environment for developing enterprise-class Web applications. Crystal Reports .NET can leverage this platform to create zero-client reporting applications that don't require anything installed on the client side. Using a viewer that has been created specifically for use with ASP .NET, you can quickly create Web-based applications that integrate complex reports, which can include tables, charts, and more.

In this chapter, we are going to look at how to integrate and view reports from within Web-based applications created with Visual Studio .NET. In addition, we will look at some of the run-time customizations that can be made to your reports, as well as some issues around Web application deployment. This will consist of:

- Determining the correct object model
- `CrystalDecisions.Web` namespace
- Using the Crystal Web Forms Viewer
- Customizing the Crystal Web Forms Viewer
- Passing information to the Web Forms Viewer

As we go through this chapter, we will be building forms for use in Web-based reporting applications, which demonstrate many of the Crystal Reports .NET features that can be used in your own Web applications.

Obtaining the Sample Files

All the example reports and code used in this chapter are available for download. The download file can be obtained from www.wrox.com. Once you have downloaded the files, place them in a folder called `Crystal.NET2003\Chapter05` on your hard drive.

In this chapter, all of the completed projects are included in the downloadable code as well as the reports used throughout the chapter, so you can either browse through the finished projects or create your own projects from scratch using the components provided.

You can use the code as you go through the chapter or cut and paste code samples into your own Web application.

Planning Your Application

If you are developing Web applications with Visual Studio .NET, chances are you are well acquainted with ASP .NET (and if you aren't, you soon will be!). ASP .NET is not really a language, per se, but rather a set of interrelated technologies and components that come together in one framework to deliver robust Web applications. As a developer, you probably already know that the most important part of creating an application is the planning and design of the application, before the coding actually starts. The integration of Crystal Reports into Web applications is no different; a little bit of planning goes a long way.

The first thing we will need to do, before we write a single line of code, is to determine what type of reports we want to deliver in our Web application and how they are going to be used. Are they listing or grouped reports? Are they used to check data entry in a form before submitting it? What will the reports look like? Will users want to print the reports from their browser or export to another format such as PDF, RTF, or Excel? All of these questions can help you gather the information you need to design your reports and get a handle on how they are going to be delivered.

Even if you don't have your own reports to work with, you can still work through this chapter; sample reports are available in `C:\Program Files\Visual Studio .NET 2003\Crystal Reports\Samples\` or in the download files for this chapter.

Once you understand the type of functionality you would like to deliver to the user, you can sit down and start planning how Crystal Reports will be integrated into your Web application. Crystal Reports .NET uses a feature-rich report viewer, available out of the box, which can be inserted onto a Web Form and used to view reports. The viewer itself has features that are similar to the Windows Form Viewer and has an extensive object model, allowing you to set the source of the report, the appearance of the viewer itself, and what happens when different events fire, among other things.

When working with Web applications, most users seem to prefer that we pop up an additional window to display reports. This allows them to have the full browser area to view the report, and we can pass properties like the report source and viewer settings to this Web Form. This allows us to use one report viewing form throughout the Web application and just set the properties we need each time.

The options for working with reports are endless. Based on users' access rights in your application, you could set a specific record selection formula or allow users to set and retain parameters they use

frequently, or even establish profiles of their favorite reports, so they can run it with all of their settings in place with one click.

Like integrating reporting into Windows applications, the report integration should be driven by the user's requirements, but how these features are delivered is up to you. As you go through the rest of the chapter, think about how the different customization features could be used in your development. If you are not at a point where you can integrate these features into your application, the various properties, methods, and events are grouped together by function to make it easier to come back and look them up.

A Brief History of Crystal Web Development

When Crystal Reports was first released, the Internet was still in its infancy, and Crystal Reports has grown right along with it. With the introduction of a Web component in Crystal Reports 7.0, based on the Print Engine already in use with its Windows development tools, developers were able to integrate reporting into their own Web applications through the use of ASP. This first implementation of Web reporting provided a powerful tool for Web developers and enabled a whole new class of reporting applications for the Web.

It wasn't long before Web developers started pushing Crystal Reports on the Web to its limit. Although version 7.0 of Crystal Reports provided a Web engine that was suitable for small workgroup applications of 5–10 users, it lacked the power to handle the first of many large enterprise Web applications that were being developed at the time.

A companion product, Seagate Info (formerly Crystal Info) was also introduced utilizing a similar framework, but adding multi-tier processing to the architecture, enabling reports to be processed on a separate machine and then viewed by the user. Unfortunately, customizing the Seagate Info user interface, or creating custom apps that accessed this technology, proved to be cumbersome, so it really didn't take off with developers.

With the release of version 8.0, the reporting technology took another massive leap forward, but some of the same limitations persisted (such as scalability and security) until the advent of Crystal Reports 8.5 and the introduction of Crystal Enterprise 8.5. Leveraging the architecture and code base from Seagate Info, Crystal Enterprise provided a robust application framework that developers could use to create applications that could be scaled from 1 to 10 to 10,000 users and beyond. With subsequent releases, Crystal Enterprise has grown in functionality to include a variety of distribution methods, including e-mail, writing to a disk or FTP site, and now features integration with LDAP, Windows Security, Active Directory, and other security providers.

So where does that leave you, the Crystal Reports .NET developer? Well, to start, you don't need to buy any additional tools or licenses to integrate reporting into your Web applications; Crystal Reports .NET provides all of the tools you need to create Web-based workgroup applications.

To deploy applications beyond a workgroup implementation of 5–10 users to a large number of users, you will need to purchase an additional license from Crystal Decisions. Also, if you need to offload processing in a true n-tier application or want to schedule or redistribute reports, you may want to consider moving your application to Crystal Enterprise. For more information on Crystal Enterprise, visit the Crystal Decisions Web site at www.crystaldecisions.com.

The other great news is that Crystal Reports .NET builds on the Web functionality found in previous products and provides a feature-rich development environment and a rich user experience for viewing reports on the Web. If you haven't looked at the Crystal Reports Web technology in a while, you are going to be pleasantly surprised.

Exploring the Development Environment

When creating ASP .NET Web applications, you don't need a specialized editor to develop the required components; you could just crack open Notepad and create all of the files required yourself. Thankfully, Visual Studio .NET provides a feature-rich development environment that makes things a bit easier when creating ASP .NET applications and there are a number of Crystal-specific components for use in Web applications.

To start, in the toolbox under the Web Forms section, you will find the `CrystalReportViewer`, which we will be working with a little later. When you drop or draw this viewer on a Web Form, you can set a number of properties and use the viewer to display a What-you-see-is-what-you-get (WYSIWYG) preview of your report.

In addition to the `CrystalReportViewer`, there is also a `ReportDocument` component available in the Components section of the toolbox. We use this component to add strongly typed and untyped reports to a form. (If you just opened this book and flipped to this chapter, you may be wondering what a typed report is; don't worry, we'll get to that a little later in the chapter.)

Finally, like most Windows applications, the majority of our report integration will take place in the code view of the form.

Using the object models provided by Crystal Reports .NET, you have almost complete control over the report's appearance and behavior.

Before You Get Started

Before we can actually get into creating Web-based applications, you will need to check and see if you have all of the required components installed to run these applications. ASP .NET Web applications run on a Web server that can either be located on your local machine or on another server that you have access to that has Internet Information Server (IIS) installed.

When you installed Visual Studio .NET, you may have received an error message if you did not have a Web server installed on your machine at that time. If you are working on a computer that does not have IIS installed and the required .NET components loaded, you will need to have access to a server that does in order to create the forms and applications demonstrated in this chapter.

For more information on installing the .NET Framework and preparing a Web server for application development, check out the Visual Studio .NET Combined Help Collection and search for "Configuring Applications."

Starting a New Web Application with VB .NET

The first thing we need to do to get started is to create a new Web application using Visual Basic .NET. Included with the download files for this chapter are a number of projects that are related to the different sections in this chapter. To walk through the examples that follow, you can either create a new solution or open the one that is provided (the same applies to the other projects; you can either follow along or create your own).

To create a new Web application, from within Visual Studio, select File → New → Project and from Visual Basic Projects, select ASP .NET Web Application and specify the URL (web_viewer_basic) and location for your project files. Because you are creating a Web application, the location will be a Web server that you have access to and the name of your project will actually be used to create a virtual directory on this server. (The good news is that Visual Studio .NET will automatically do this for you if you are building the application from scratch; there is no need to create the folder and virtual directory prior to creating a new project.)

If, however, you choose to use the supplied download code, you should create a virtual directory (in our case, this is C:\Crystal .NET2003\Chapter05\web_viewer_basic) by selecting Control Panel → Administrative Tools → Internet Information Services, and then right-clicking Default Web Site. This will open another menu, shown in Figure 5-1.

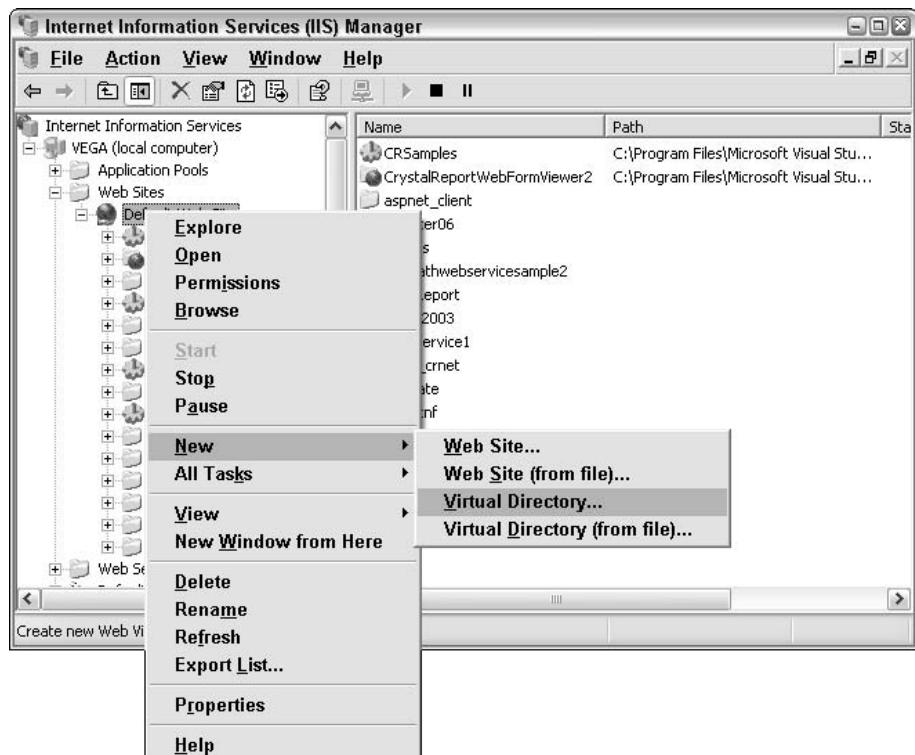


Figure 5-1

Select New → Virtual Directory, and the virtual directory wizard will commence. Assign the new directory the alias `web_viewer_basic` and set the path to `C:\Crystal.NET2003\Chapter05\web_viewer_basic`. Make sure both read and write are enabled and finish the wizard.

Either way you choose to do it, the development environment will open with a default form that we will be using in the section. Throughout the chapter, we will be using only one or two Web Forms to demonstrate different integration features, but the same concepts can be applied to your own Web applications.

Before you go any further, we need to get some basic architecture decisions for your Web application out of the way, starting with a brief discussion of the object models available within Crystal Reports .NET.

Determining the Correct Object Model

When working with Web applications, there are two different object models to choose from, each with its own capabilities and strengths. The first, contained within the *Crystal Reports Web Forms Viewer* object model (`CrystalDecisions.web`), contains all of the functionality required to view a report in the Crystal Reports Web Forms Viewer, including the ability to set database logon information, pass parameters and record selection, control the viewer's appearance, and view reports, including reports consumed from an XML Report Web Service.

Using the `CrystalDecisions.Web` object model, you are covered for most basic report integration requirements, but you have no control over the report itself at run time. You won't be able to change the record selection for any subreports that appear in your report and you won't have access to modify report elements, like groups and sorting, or formula fields.

For complete control over the report and its content, you need to use the *Crystal Reports Engine* object model (`CrystalDecisions.CrystalReports.Engine`) in conjunction with the viewer object model. This will allow you complete control over your report and the objects and features contained within. Using the Crystal Reports Engine means that you have a rich object model that can be used to modify even the tiniest elements of your report.

You will also need to use the Report Engine object model if you are using ADO (.NET or Classic ADO) as the data source for your report (covered in Chapter 7, "Working with .NET Data").

It is important to note that the Crystal Reports Engine object model cannot stand alone; it provides no way to view a report and relies on the Crystal Reports Web (or Windows) Forms Viewer to actually view the report.

Crystal Decisions recommends that you do not overlap the two object models and try to use properties and methods from both at the same time. An example would be where you are setting a parameter field value in the Report Engine object model; you wouldn't want to also try to set a parameter field in the same report using the Crystal Reports Windows Forms Viewer object model. Try to pick an object model based on your requirements and (as I recommended in Chapter 4, "Report Integration for Windows-Based Applications," with the Windows Forms Viewer) stick with it!

Understanding the CrystalDecisions.Web Namespace

The `CrystalDecisions.Web` namespace contains all of the classes that relate to functions available within the Web Forms Viewer itself. The following table illustrates the different classes that are available, as well as their use in Web applications.

Class	Description
<code>CrystalReportViewer</code>	Contains the properties, methods, and events relating to the <code>CrystalReportViewer</code> and viewing reports. Note: some properties of this class are inherited from <code>CrystalReportViewerBase</code> .
<code>CrystalReportViewerBase</code>	Contains properties for setting the target browser edition, database logon information, and so on.
<code>DrillEventArgs</code>	Provides data for the <code>Drill</code> event on main reports and subreports. <code>Drill</code> events fire when a user drills down into a group or summary on a particular main report or subreport.
<code>DrillSubreportEventArgs</code>	Provides data for the <code>DrillDownSubreport</code> event on main reports and subreports. <code>Drill</code> events fire when a user drills down into a group or summary on a particular main report or subreport.
<code>NavigateEventArgs</code>	Provides data for the <code>Navigate</code> event. When a user navigates through the pages of a report, the <code>Navigate</code> event fires each time. This can be used to notify the users when they have reached the last page, to call custom actions, and so on.
<code>SearchEventArgs</code>	Provides data for the <code>Search</code> event. The <code>CrystalReportViewer</code> includes an integrated search function to search for values within a report. The <code>Search</code> event fires when a user searches for a value and could be used to trigger a search of another report or other report descriptions, and so on.
<code>ViewerEventArgs</code>	Provides data for the <code>Viewer</code> event. The <code>Viewer</code> event fires when some action has occurred within the viewer itself and can be used to launch other actions when the viewer loads, and so on.
<code>ZoomEventArgs</code>	Provides data for the <code>ViewZoom</code> event. The <code>ViewZoom</code> event fires when the zoom is changed on the viewer and can be used to suggest the best resolution for a particular report (100%, 200%, and so on), or if you are showing two reports in viewers side-by-side, to synchronize the zoom factor between the two (so the magnification on both reports stays the same; if you change one to 200%, the other view changes as well).

Using the Crystal Report Viewer for Web Forms

For report files that live externally to your application (for instance, as a standalone report file, created with either this or a previous version of Crystal Reports), there is not much to creating a simple preview form for your report. We are going to walk through that process in the following section.

Earlier we created a new project called `web_viewer_basic` and within that project there should be a default Web Form (`WebForm1.aspx`) that was created when you created the project. To start, we need to drag or draw the Crystal Report Viewer onto our Web Form, as shown in Figure 5-2.

From that point, we need to set the `ReportSource` property to let the viewer know where to get the report. To access this property, locate the Properties window for the Crystal Report Viewer and open the `(DataBindings)` property, by clicking the ellipse at the side, to show the dialog that appears in Figure 5-3.

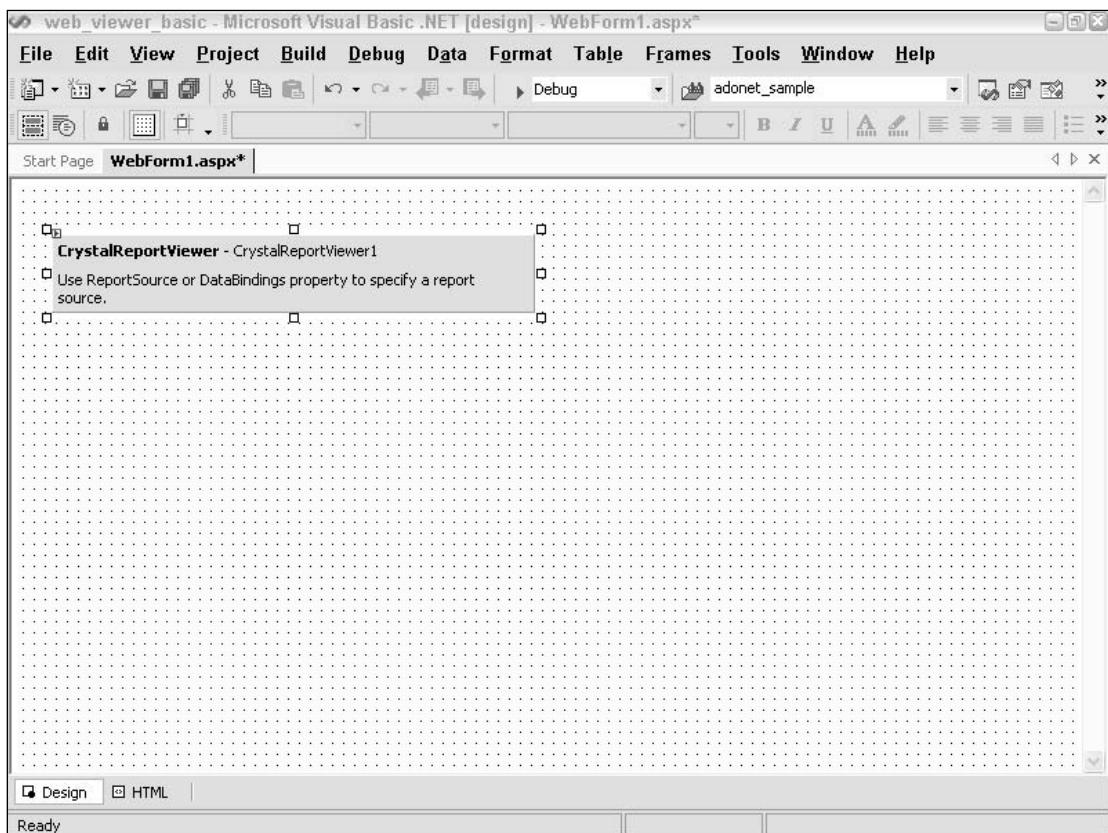


Figure 5-2

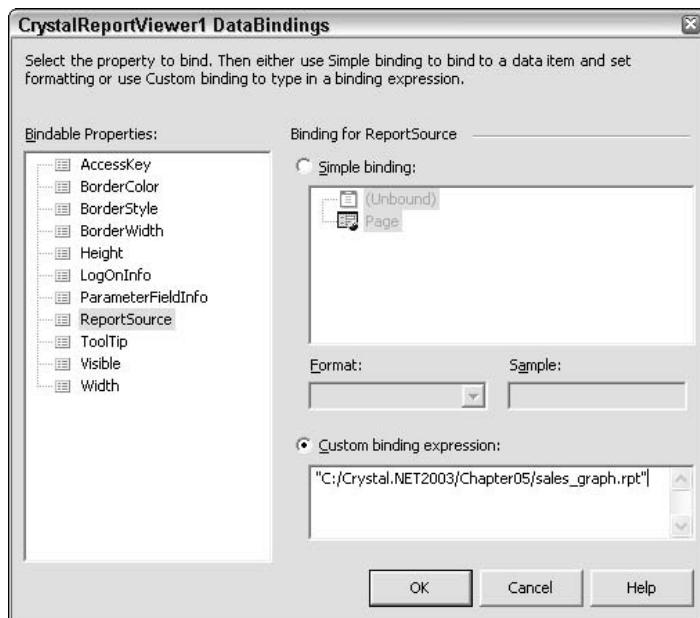


Figure 5-3

Click the `ReportSource` property and click the radio button to select Custom Binding Expression. In this example, we are going to assume that you have unzipped the download files for this chapter to your hard drive in a folder called `c:\Crystal.NET2003\Chapter05`. Included in these files is a Sales Graph Report (`sales_graph.rpt`) that we will be using in this walkthrough.

Once you have entered the name and path for your report, click OK to accept your changes and return to the form we were working with. The report will now be displayed in a Preview mode in the form designer, as shown in Figure 5-4.

If we were working with a Windows Form, this would be all that is required to actually preview a report. Because Web Forms work a little differently, there is an extra step involved before we can run our application and preview our report.

Chapter 5

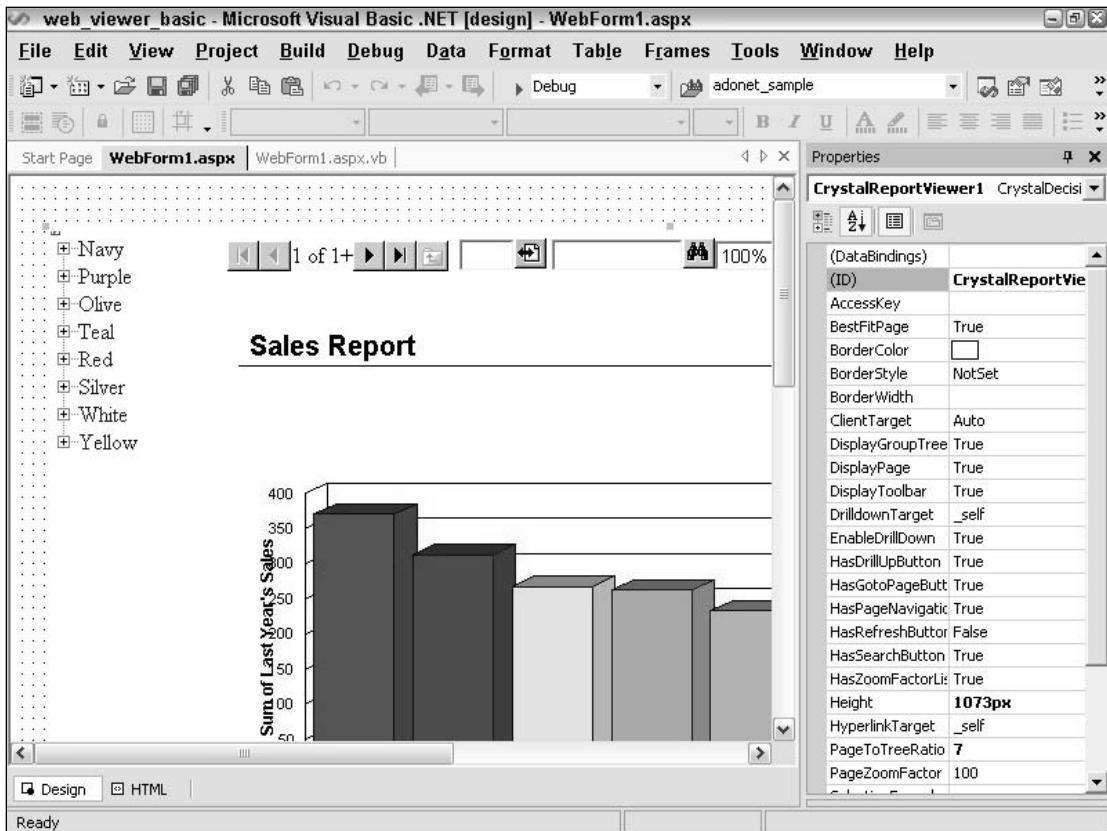


Figure 5-4

Double-click anywhere on your form to open the code view for the form and locate the section of code marked Web Form Designer generated code. Expand this section to find the `Page_Init` function and add a line of code immediately after the `InitializeComponent()` call to bind your report to the viewer when the page is initialized:

```
Private Sub Page_Init(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles MyBase.Init  
    'CODEGEN: This method call is required by the Web Form Designer  
    'Do not modify it using the code editor.  
    InitializeComponent()  
    CrystalReportViewer1.DataBind()  
End Sub
```

Whenever you run your application and preview the form, your report will be displayed in the Crystal Report Viewer, as shown in Figure 5-5.

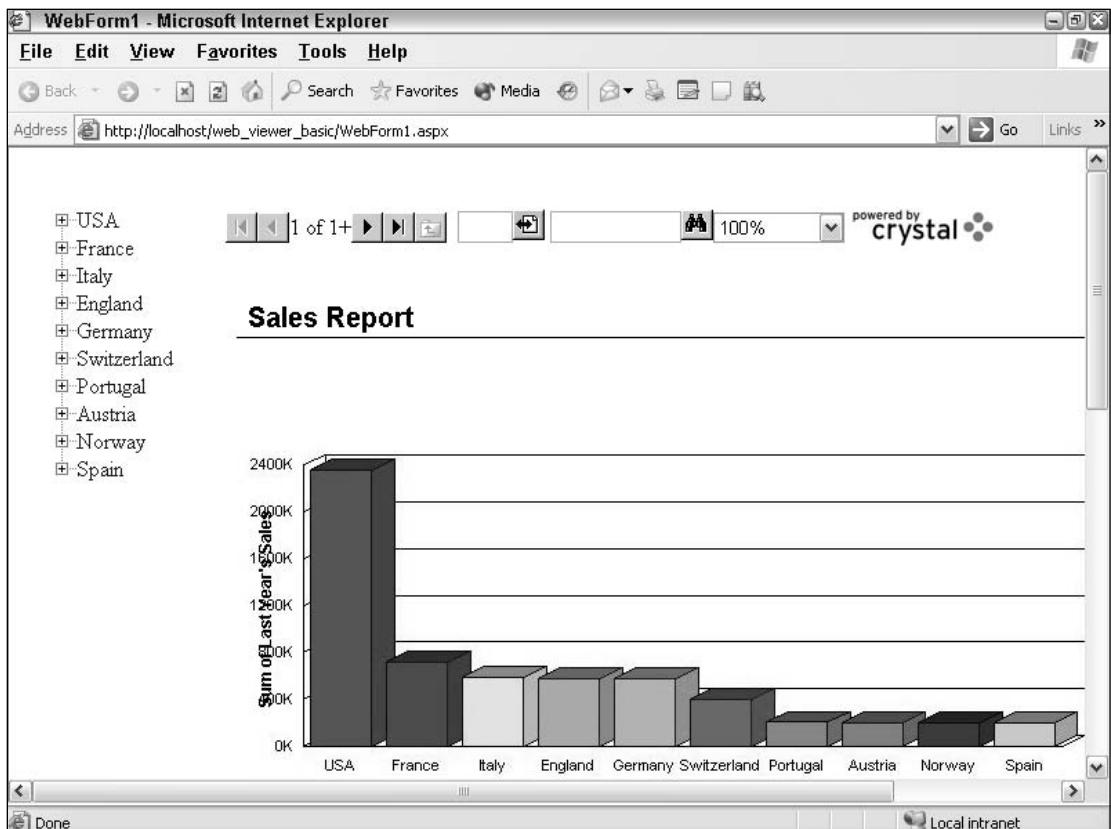


Figure 5-5

The viewer interacts with the Crystal Reports Print Engine, runs the report, and displays the results. From your report preview, you can drill-down into the details or search for a value, without having to do any additional coding.

If you have only one or two reports that you want to integrate into a view-only application and you don't need to customize any features at run time, this may be all you need. But for applications that required a more sophisticated integration with Crystal Reports .NET, you probably need to look a bit further.

In the following sections, we are going to walk through adding a report to your application, binding the report to the Crystal Report Viewer, and customizing the viewer.

Adding a Report to Your Application

To add a new report to your application, you have two choices: You can either use an existing report that you have created (using this or a previous version of Crystal Reports), or you can use the Report Designer integrated within Visual Studio .NET to create a report from scratch. For our purposes, we are

Chapter 5

going to add an existing report to our next sample application (for more information on creating reports from scratch, check out Chapter 2, “Getting Started with Crystal Reports .NET”).

In this example, we will add the Sales Graph Report to `web_viewer_basic2`. Whereas the approach in our first example favors publication of a report that will always be found by the same path but is subject to regular updates, this option favors a report that is not likely to change in structure and therefore can be incorporated into the application, allowing us to alter its features, or even build it from the ground up. This is also relevant to whether we are using strongly typed or untyped reports, as we discussed in Chapter 4.

Once again, you have the choice of building the project or using the code provided, but remember that if you use the code provided, you must create a virtual directory for it in IIS on your machine.

To add our Sales Graph Report to this new project, select Project → Add Existing Item, which will open the dialog shown in Figure 5-6. Change the drop-down list to show All Files and specify `*.rpt` for the file name to filter the list to show only the available reports.

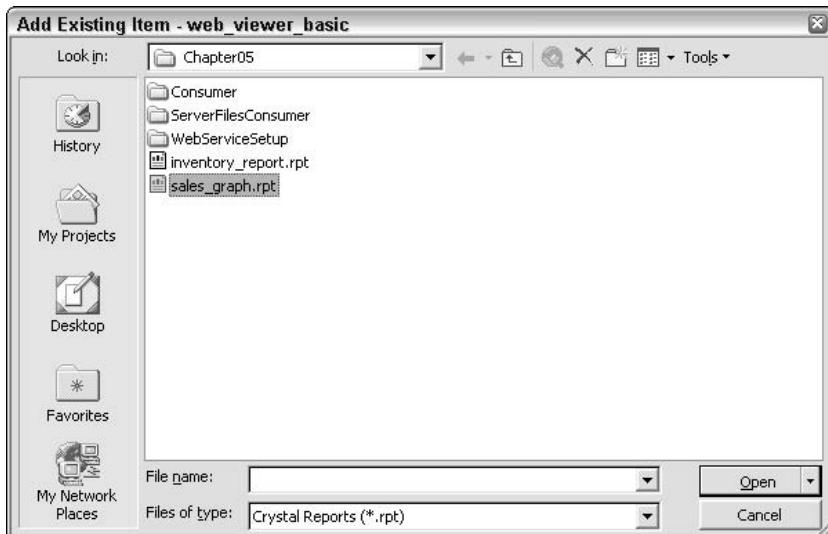


Figure 5-6

Once you have selected the `Sales_Graph.rpt` report, click Open and the report will be added to your project in the Solution Explorer, as shown in Figure 5-7.

Once you have added your report to your project, it will appear in the Solutions Explorer and you can view the design of the report using the Report Designer and access its properties through its Properties page (Figure 5-8).

Report Integration for Web-Based Applications

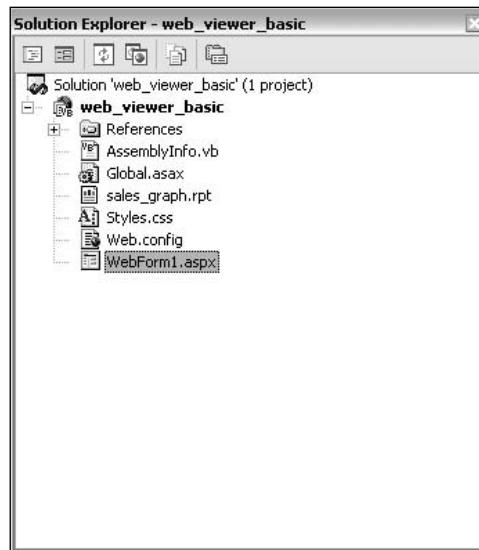


Figure 5-7

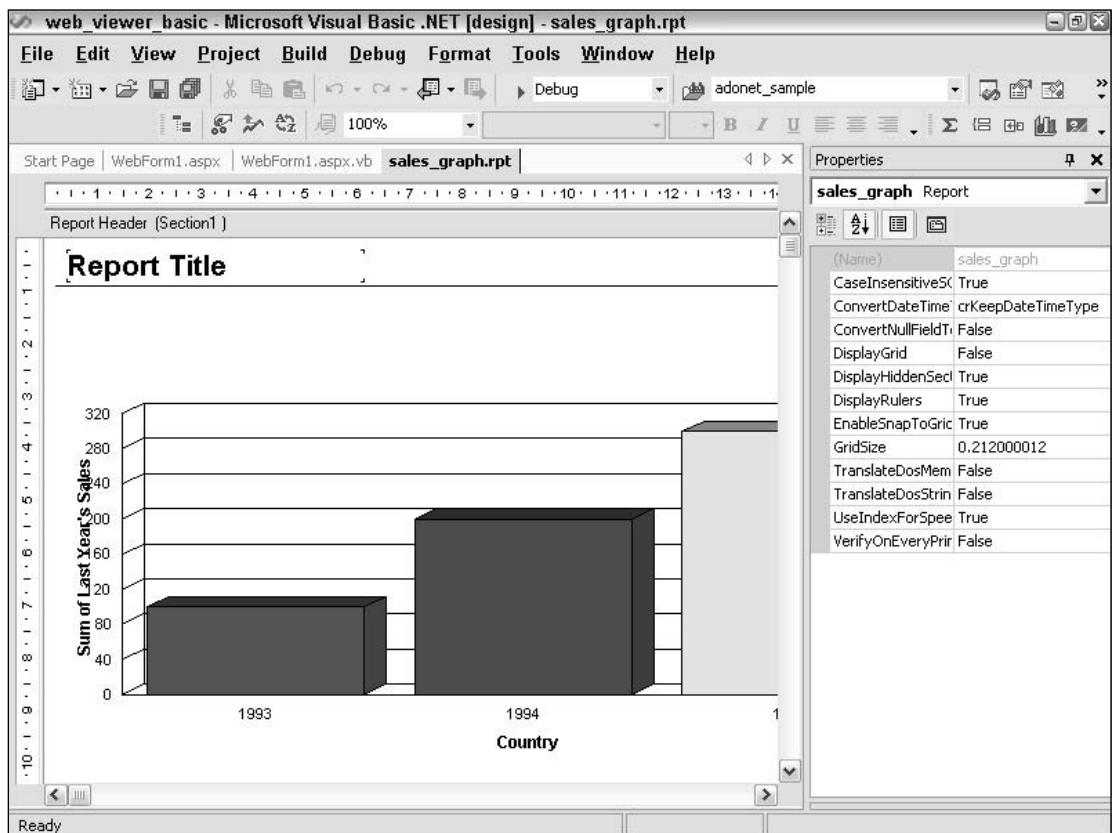


Figure 5-8

Adding the Report Viewer to a Web Form

You can add the Crystal Report Viewer from the Web Forms Toolbox and drag or draw the viewer onto your form. Unlike the Crystal Report Viewer for Windows Forms, the Web Forms Viewer does not provide a view of how the viewer will appear on your page until you actually bind a report to it, at design or run time.

In fact, we'll do this now for `web_viewer_basic2`. Just drag a `CrystalReportViewer` over from the Toolbox and place it on the Web Form. Don't bother setting a report source for it yet, as we are about to look at different methods for binding the report to the viewer.

The Crystal Report Viewer can be used on existing forms to display a report side-by-side with other controls, or you could add the report to a new form to have a separate window for previewing reports.

Binding a Report to the Report Viewer

With the Report Viewer added to your form, you now need to bind a report to the viewer itself. As we saw in Chapter 4, "Report Integration for Windows-Based Applications," there are five different ways to bind a report to the Crystal Report Viewer:

- By report name
- By report object
- By binding to an untyped report
- By binding to a strongly typed report
- By binding to a strongly typed cached report

Binding by Report Name

To bind a report using the report name, as we did in our first example, all you need to do is set the `ReportSource` property, either through the Data Bindings properties for the report or through the form's code, as shown here:

```
CrystalReportViewer1.ReportSource = "C:\Crystal.NET2003\Chapter05\web_viewer_basic\sales_graph.rpt"
```

Then in the `Page_Init` event of your form, you would need to add a single line of code to call the `DataBind` method immediately after the `InitializeComponent()` call, as shown here:

```
CrystalReportViewer1.DataBind()
```

If you prefer to set the initial report source using the property pages, you will need to open the Properties page for the report viewer, and then select the `(DataBindings)` property as we did in the first example.

But even as we have added the report to our application, we could also bind the report by creating a report object, loading the report into the object, and binding the object to the Crystal Report Viewer.

Binding by Report Object

Binding by a report object works slightly differently depending on if you have added the report to your project, or if you are referencing it externally. For a report that resides external to your application, you need to first specify an import of the `CrystalDecisions.CrystalReports.Engine`, which will allow you to create the object.

If this is not already present, we shall add it. In the Solution Explorer, right-click your project title and select Properties from the right-click menu to open the dialog shown in Figure 5-9.

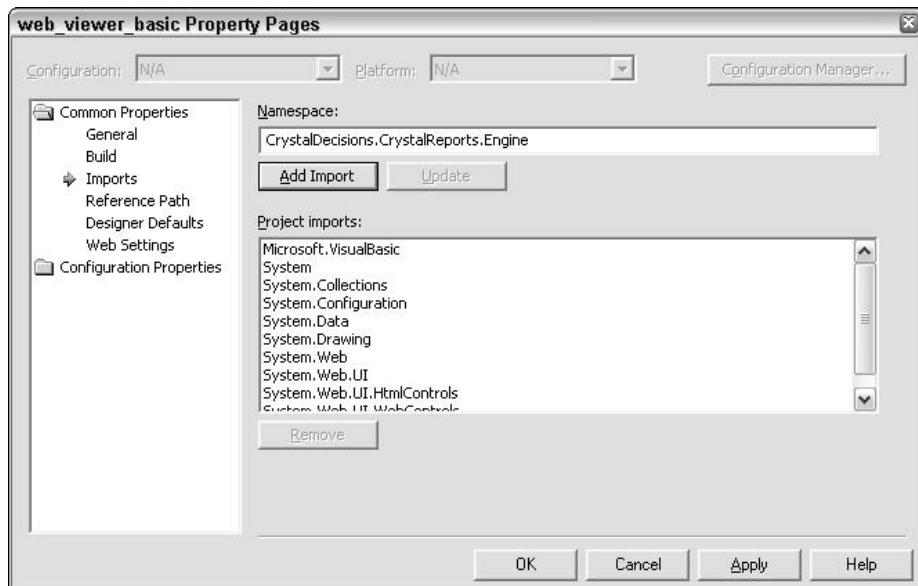


Figure 5-9

Under the Common Properties folder, use the Imports option to add the `CrystalDecisions.CrystalReports.Engine` namespace to your project and click OK to return to your form.

With this namespace imported, we now need to create the report object as a `ReportDocument` in the form's Declarations section, as shown:

```
Dim myReport as New ReportDocument()
```

With our object now ready to be used, we can load the external report file by placing the following code under the `Page_Init` event:

```
...
InitializeComponent()
myReport.Load("C:\Crystal.NET2003\Chapter05\sales_graph.rpt")
CrystalReportViewer1.ReportSource = myReport
```

If the report you are using has actually been added to your project (as ours was earlier), a class was automatically generated for the report, so we could use the class instead to bind the report to the viewer (though the import and Public declaration remain the same):

```
...
InitializeComponent()
myReport = New sales_graph()
CrystalReportViewer1.ReportSource = myReport
```

Or, if you wanted to eliminate the myReport variable:

```
CrystalReportViewer1.ReportSource = New sales_graph()
```

Which method you choose will depend on where the report is physically located and how you wish to access it.

Binding to an Untyped Report

When working with Crystal Reports .NET, you can add individual report files to your project and use and reference them to view reports from your application. Taking this a step further, you could also use these reports as components, which is where we start looking at typing.

When integrating reports into an application, we can use either strongly typed reports or untyped reports. If you have been working with Visual Studio .NET for any length of time, you have probably heard of strongly typed objects. A strongly typed object is predefined with a number of attributes that are specific to that object, giving programmers more structure and a rigorous set of rules to follow, whereas an untyped report does not have this structure or rules applied to it, making it more difficult to work with.

Within the frame of reference of Crystal Reports .NET, a strongly typed report can be any report that has been added to your project. When you add a report to a project, you will notice that in addition to the report, another file (with a .vb extension) is also added, as shown in Figure 5-10.

This file is hidden until you select Show All Files from the Solution Explorer.

This additional file is the report source file and contains a report class specific to each report, called `ReportDocument`, which is derived from the `ReportClass` class and is created automatically for you.

An example of an untyped report would be a report that is stored externally to your project. For instance, you could view a report by setting an external reference (as in our first example, where we set the `ReportSource` property in the `(DataBindings)` to "C:\Crystal.NET2003\Chapter05\web_viewer_basic\sales_graph.rpt").

We'll just have a brief look at how we do this. Create a virtual directory called `web_viewer_untyped` pointing at C:\Crystal.NET2003\Chapter05\web_viewer_untyped (if this is where you have downloaded the source code to), or alternatively, build it from scratch.

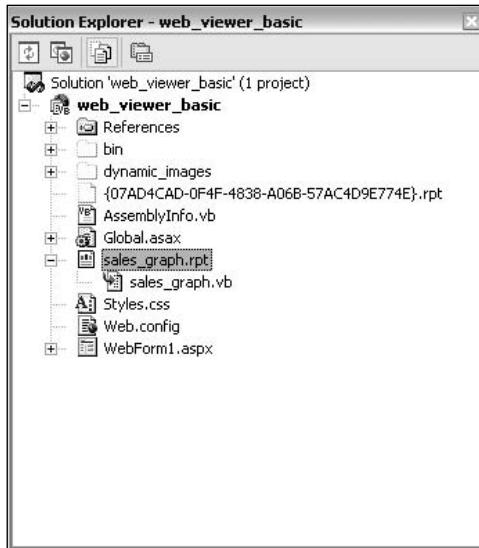


Figure 5-10

To add a report component to your application, switch to the Layout view of your form and look in the toolbox under Components. In this section, you should see a component labeled `ReportDocument`. Drag this component onto your form, which will open the dialog shown in Figure 5-11.

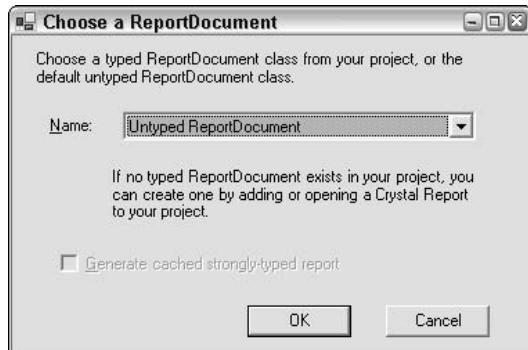


Figure 5-11

It is in this dialog that you set whether your `ReportDocument` component is typed or untyped. If you select `Untyped ReportDocument`, you are not really accomplishing much new here. Drag a new `CrystalReportViewer` onto the Web Form, and then load a report into `ReportDocument1` and bind the component to the viewer.

```
...
InitializeComponent()
Dim ReportDocument1 As New ReportDocument()
ReportDocument1.Load("C:\Crystal.NET2003\Chapter05\sales_graph.rpt")
CrystalReportViewer1.ReportSource = ReportDocument1
```

Binding to a Strongly Typed Report

Finally, you can choose to add a strongly typed report component, which probably has the simplest binding method of all. First of all, add the report that you wish to bind to your project. In our case, this will be `sales_graph.rpt`. To create a strongly typed `ReportDocument` component, drag the `ReportDocument` component onto your Web Form. (The code for this Web Form is available in the code download for the chapter as `web_viewer_stronglytyped`.) This will open the dialog shown in Figure 5-12 and allow you to create a strongly typed report.

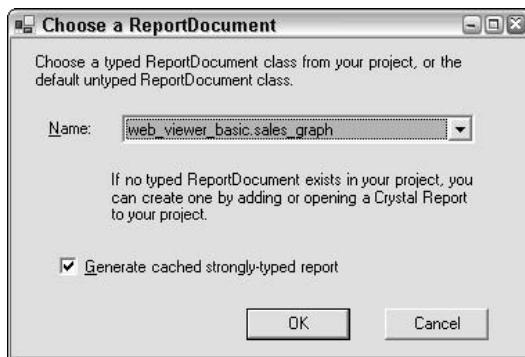


Figure 5-12

You will then see the same dialog as before, with a drop-down list of all of the available reports that are in your project. Select an existing report to create a strongly typed `ReportDocument`. Now, insert the `CrystalDecisions.CrystalReports.Engine` namespace into the project using the Properties page, as we did previously, drag on a `CrystalReportViewer`, and we're set. From that point, we just need to set the `ReportSource` property in the `Page_Init` event once more:

```
CrystalReportViewer1.ReportSource = sales_graph1
```

(where `sales_graph1` is the name automatically assigned to the `ReportDocument` component when you added it to your form).

Binding to a Strongly Typed Cached Report

Another option for strongly typed reports is the ability to use ASP .NET caching with your report. When you added your report to your application, you may have noticed that there was a checkbox for Generate cached strongly typed report. Report caching is based on the underlying ASP .NET caching model and provides an easy way to improve your application's performance.

When a report is cached, the subsequent Web Form that is used to view the report will load faster when accessed by different users. To add a report to your Web Form as a cached report, select this option as you add a strongly typed report to your application from the dialog shown in Figure 5-13.

You will see that your report file will be inserted as `cached_sales_graph1` and an additional object will be inserted into the Web Form's source file.

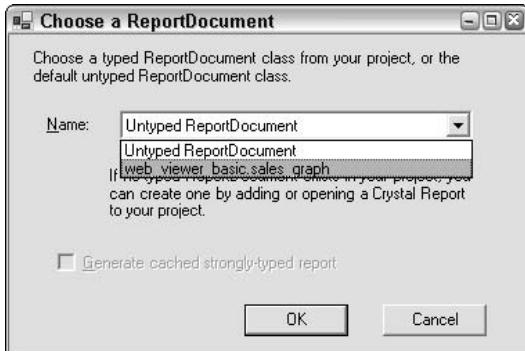


Figure 5-13

You could then bind to this particular cached report just as you would to any other strongly typed report, but with a different name:

```
CrystalReportViewer1.ReportSource = cached_sales_graph1
```

When multiple users visit the same report, they will actually be looking at a cached copy and not hitting the database in real time. (To ensure this is true, the viewer by default does not have a refresh button showing.)

So regardless of which method you choose to bind your report to the viewer, the result is the same. For ease of use and functionality provided, the easiest method is going to be to stick with strongly typed reports, because in the long run, the structure and coding standards will mean you can create reporting applications quickly, in a consistent manner.

After binding, you can run your application and when the form is loaded, the Crystal Report Viewer will run the report you have set in the `ReportSource` property and display a preview of it. Before we can move on to customizing the viewer, however, we need to look at working with secured databases.

Before we finish up with viewer basics and binding, keep in mind that reports that were created from a secure data source may require a user name and password.

To add `ch5_worldsales_northwind.rpt` to your project, select Project → Add New Item and browse for the report where you unzipped the downloaded sample files. Add it to the project and click the Open button.

Now drag and drop a `CrystalReportViewer` onto your form. For the sake of attractiveness, set the `Dock` property of the form to Fill.

You will also need to add this report as a component to your form. Switch to the Layout view of your form and look in the toolbox under Components. In this section, you should see a component labeled `ReportDocument`. Drag this component onto your form.

From the drop-down list, select the `ch5_worldsales_Northwind` report and click OK. We are now ready to get on with setting the database credentials for this report.

Chapter 5

This report has been written from a single table, `Customer`, for which we are setting the `ConnectionString`. The name of our server is `localhost`, off of the Northwind database, with `sa` as the user ID, and no password.

Add this code to the `Form1_Load` method:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
    Dim myReport = New ch5_worldsales_northwind()
    Dim myTableLogonInfos = New CrystalDecisions.Shared.TableLogOnInfos()
    Dim myTableLogonInfo = New CrystalDecisions.Shared.TableLogOnInfo()
    Dim myConnectionString = New CrystalDecisions.Shared.ConnectionString()

    With myConnectionString
        .ServerName = "localhost"
        .DatabaseName = "Northwind"
        .UserID = "sa"
        .Password = ""
    End With

    myTableLogonInfo.ConnectionString = myConnectionString
    myTableLogonInfo.TableName = "customers"
    myTableLogonInfos.Add(myTableLogonInfo)
    CrystalReportViewer1.LogOnInfo = myTableLogonInfos
    CrystalReportViewer1.ReportSource = myReport
End Sub
```

Make sure that when you are finished setting the `ConnectionString` for the table, you specify the table name you are working with prior to using the `Add` method; otherwise you will receive an error message.

Compile and run the example. The report should appear, looking identical to the previous examples.

This is a very simple example, as our report has only one table to worry about. If you have a report that features multiple tables or if you don't know the names of the tables, you could also set up a loop to go through each `report.database.table` in `report.database.tables` and set the `ConnectionString` properties for each.

In order to get all of the tables in your report and loop through them, you will need to use the Report Engine, which is covered in Chapter 9, "Working with the Crystal Reports Engine."

Setting Report Record Selection

Just like the functionality found in the Crystal Viewer for Windows applications, we can also set the record selection formula that is used to filter our report records when viewing a report through the Web Viewer.

Just like the Windows Viewer, the `SelectionFormula` property gives us the ability to return or set this value at run time. To return a record selection formula, we would simply request the property as a string, as shown here:

```
CrystalReportViewer1.SelectionFormula.ToString
```

Using the report and viewer we have been working with, we could set the record selection property when the form loads, before the call to the database, ensuring that only records for customers in the region of NC (North Carolina) are shown:

```
Dim myConnectionInfo = New CrystalDecisions.Shared.ConnectionInfo()
CrystalReportViewer1.SelectionFormula = "{Customer.Region} = 'NC'"
With myConnectionInfo
    ...
End With
```

This is almost exactly like the method you would use to change the record selection formula in a Windows-based application, so it shouldn't be too hard.

Working with Parameter Fields

Another report feature that is often used is parameter fields; a Crystal Report can use parameter fields to prompt the user for values that can be displayed on the report, used in record selection criteria, and so on. If you have played with this feature in conjunction with Windows applications, you are probably already familiar with the default dialog that appears when you run or preview a report that has parameters. In the last chapter, we looked at how to create your own dialogs to capture these parameters, noting you could always just let Crystal Reports pop the default parameter dialog and save you the trouble of creating one.

With the Crystal Viewer for Web-based applications, this is not an option; in order to use parameters with a report, we will need to create our own user interface to capture those parameters and then send them to the viewer to be processed and shown on our report. So in the following section, we are going to look at how to programmatically set different types of parameters found in your reports using the Crystal Viewer for Web applications.

Again, to walk through the examples in this section, you will need to have a report that includes parameters and make sure that the parameters are used in the report—either displayed on the report, used in a formula, in the record selection, and so on—because we want to see the results when we actually set these parameter values.

To see this in action, we are going to add yet another new project and add it to our existing one. Click File → New → Project, and select ASP .NET Web Application. Call the project `viewer_parameters` and save it to `Crystal.NET2003\Chapter05`. Make sure that the radio button Add to Solution is selected as opposed to Close Solution, as shown in Figure 5-14.

The sample report that goes along with this section is called `ch5_parameters.rpt` and to add this report to your project, select Project → Add Existing Item and browse for the report where you unzipped the downloaded sample files. Add it to the project and click the Open button.

Now drag and drop a `CrystalReportViewer` onto your form and add the report that was just added as a component to your Form. Look in the toolbox under Components. In this section, you should see a component labeled `ReportDocument`. Drag this component onto your form.

From the drop-down list, select the `ch5_parameters.rpt` report and click OK.

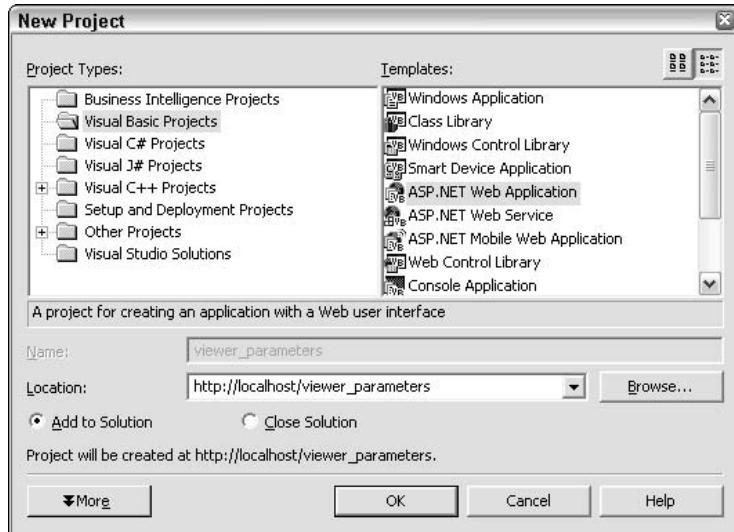


Figure 5-14

Then in the `Page_Init` event of your form, add a single line of code to call the `DataBind` method immediately after the `InitializeComponent()` call, as shown here:

```
CrystalReportViewer1.DataBind()
```

If you prefer to set the initial report source using the property pages, you will need to open the Properties page for the Report Viewer, and then select the `(DataBindings)` property, as we did in the first example. Then right-click the project name, and click Set As Startup Project. You're now ready to start setting working with parameters.

Again, a good starting place for working with parameters is looking at the different types of parameters you could have in a report; in addition to a parameter's data type (String, Number, and so on), you can also set options for how the parameter is entered. These options will seem familiar to you if you have already read the previous chapter, but they are repeated here just to drive the point home and for the benefit of anyone who doesn't want to flip back and find that particular section. There are three basic options available when creating parameter fields:

- Discrete* parameters accept single, discrete values; depending on how the parameter field was set up, it can accept only one value or multiple values.
- Ranged* parameters can accept a lower and upper value, selecting everything that is within that range of numbers, letters, and so on.
- Discrete and Ranged* parameters support a combination of the two different types, where you can enter multiple discrete values as well as multiple ranges.

Parameters within a report are contained within a collection called `ParameterFields` and within the collection, a set of attributes is stored for each field using the `Parameter Field` object. As we walk through

Report Integration for Web-Based Applications

In this example, we will set properties relating to this object in this collection and then use this collection to set the `ParameterFieldInfo` property of the viewer, as shown subsequently:

```
crystalReportViewer1.ParameterFieldInfo = myParameterFields
```

To set the values for a particular parameter, we first need to dimension both the collection and object. We also need to create a variable to hold the value that we want to pass to the report for this parameter.

Because in this example we are passing only a discrete value to the parameter, we would simply create a variable by dimensioning it as a `ParameterDiscreteValue`. All we need to do at that point is set the value of the parameter equal to some value we have entered or hard-coded and then set the `ParameterFieldInfo` property of the Report Viewer.

Because we want to enter the value ourselves, we would want to put a textbox and a command button on another form to collect and submit the value to be used in our parameter to the form with the viewer present. To create a new form, select Project → Add Web Form and then add a textbox, label, and command button, which would make the form look something like Figure 5-15.

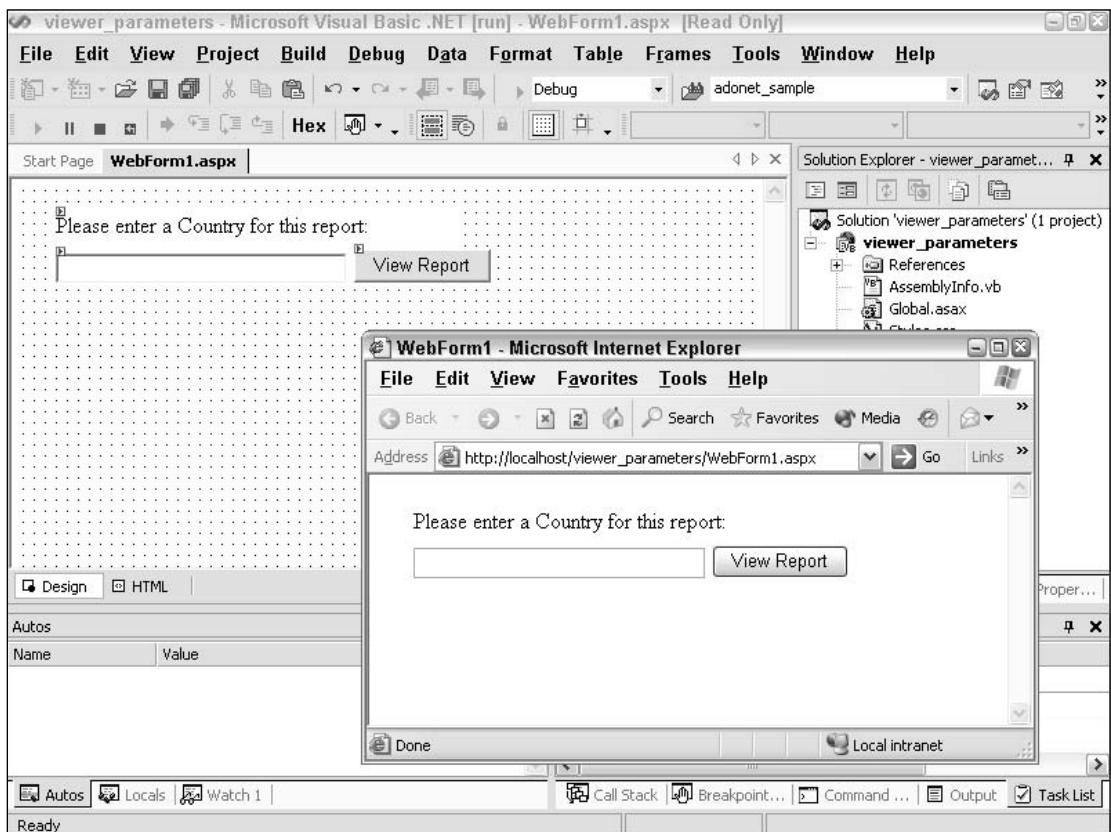


Figure 5-15

Chapter 5

Set the properties of the textbox to make it blank when the form appears and change the name of the command button to “View Report.” Then put the code behind to use Response.Redirect to go to the form that is hosting your Crystal Viewer and pass the value entered in the textbox as a URL parameter.

You can now go to the page where your viewer resides and use the following code to set your parameter fields. In this example, we are using a variable called “PassedInValue” that contains the value passed from the previous page.

```
...
Dim myParameterFields As New ParameterFields()
Dim myParameterField As New ParameterField()
Dim myDiscreteValue As New ParameterDiscreteValue()

myParameterField.ParameterFieldName = "OriginCountry"
myDiscreteValue.Value = PassedInValue
myParameterField.CurrentValues.Add(myDiscreteValue)

myParameterFields.Add(myParameterField)

crystalReportViewer1.ParameterFieldInfo = myParameterFields
crystalReportViewer1.RefreshReport
```

...

The next scenario we are going to look at is for multiple discrete variables; in this situation we would use the same methods, only instead of setting a single value using the ParameterDiscreteValue variable, we would continually redeclare this variable and use it to push values into the ParameterField object. Again, the same method would apply here—creating a front-end page to collect the parameter information and then passing it to the page with the viewer on it. This would go behind the viewer page and again you would have to pass the values from the previous page. Here we have hard-coded them to make this code a bit easier to follow.

```
...
Dim myParameterFields As New ParameterFields()
Dim myParameterField As New ParameterField()
Dim myDiscreteValue As New ParameterDiscreteValue()

myParameterField.ParameterFieldName = "OriginCountry"

myDiscreteValue.Value = "USA"
myParameterField.CurrentValues.Add(myDiscreteValue)

myDiscreteValue= New ParameterDiscreteValue()

myDiscreteValue.Value = "Canada"
myParameterField.CurrentValues.Add(myDiscreteValue)

myParameterFields.Add(myParameterField)

crystalReportViewer1.ParameterFieldInfo = myParameterFields
...
```

In this example, we have used the variable twice to push both “USA” and “Canada” to the report. Remember, in your actual report design you must specify the parameter option—for example, whether it is a discrete, multi-value discrete, or ranged parameter. Otherwise you will receive an error message when running your report.

Finally, our last type of parameter is a ranged parameter, which can be set using the same method except instead of a discrete value, we are going to pass in a range, with a start and end value, as shown in the following code:

```
...  
  
Dim myParameterFields As New ParameterFields()  
Dim myParameterField As New ParameterField()  
Dim myDiscreteValue As New ParameterDiscreteValue()  
Dim myRangeValues as New ParameterRangeValues()  
  
myParameterField = New ParameterField()  
myParameterField.ParameterFieldName = "LastYearsSales"  
  
myRangeValues.StartValue = 100000  
myRangeValues.EndValue = 500000  
  
myParameterField.CurrentValues.Add(myRangeValue)  
myParameterFields.Add(myParameterField)  
  
...
```

You can also combine the two methods, depending on the type and number of parameters you have created in your report. You could have two parameters, for example, that accept discrete values, one that accepts a range, and still another that accepts a combination of the two. The methods used to set the values for these parameters are the same, so users can easily enter parameter values through your own application and filter the report content.

Customizing the Appearance and Layout of the Report Viewer

The `CrystalReportViewer` class contains all of the properties, methods, and events that relate to the viewer itself—its appearance, methods that are used to make the viewer perform certain actions (such as refresh or go to the next page), and events that can be used to determine when a particular event (such as drill-down or refresh) has occurred. To start learning how to work with the viewer, we are going to start with the basic properties and move on from there.

To get started, we need to create a new project to work in. From within Visual Studio, select File → New → Project and from within Visual Basic Projects, select ASP .NET Web Application and specify a name and location for your project files, as shown in Figure 5-16.

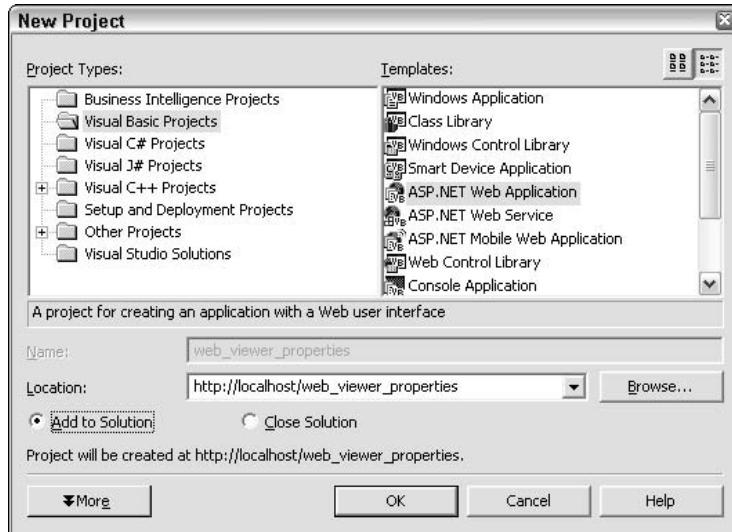


Figure 5-16

In the sample files, we have called this project (`web_viewer_properties`). Once you have selected a name for your project and clicked OK, the development environment will open with a default form that we will use in the section. Alternatively, you can create a virtual directory for this project using the sample code provided.

We also need to add a report to work with in this section, so select `Project → Add Existing Item` and select `product_listing_bytype.rpt`. Add this to your project, insert the `CrystalDecisions.CrystalReports.Engine` namespace, drag across a `ReportDocument`, and place this on the form (selecting `product_listing_bytype.rpt` out of the options on the dialog), and then insert the `CrystalReportViewer` and set the binding to this report in the `Page_Init` event:

```
CrystalReportViewer1.ReportSource = product_listing_bytype1
```

You are now ready to get started!

When you were working through the earlier example, binding to a viewer and previewing your report, you may have noticed that there is a standard set of icons and layout that appears by default on the `CrystalReportViewer`. You can control most of the aspects of the viewer and toolbar by setting a few simple properties, as shown in Figure 5-17.

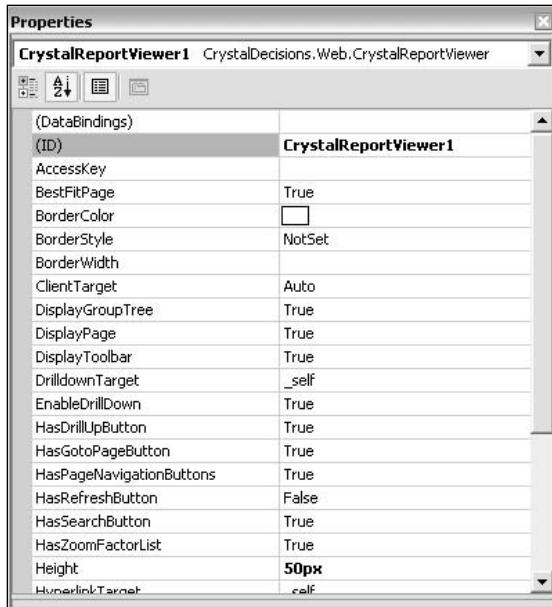


Figure 5-17

The area at the top of the viewer is the toolbar, which can be shown or hidden as an entire object or you can choose to show only certain icons. On the left-hand side is a Group Tree, generated by the grouping that you have inserted into your report. The properties that control these general properties are Boolean and are listed below:

Property	Description
BestFitPage	For showing the report as-is or with scroll-bars
DisplayGroupTree	For showing the Group Tree on the left-hand side of the viewer
DisplayPage	For showing the page view
DisplayToolbar	For showing the entire toolbar at the top of the viewer
SeparatePages	For displaying a report in separate pages or one long page

All of these properties default to True, and you cannot change the position of any of these elements; they are fixed in place on the viewer. You can, however, hide all of these icons and create your own buttons for printing, page navigation, and so on.

For the icons within the toolbar, you can also set simple Boolean properties to show or hide a particular icon, as shown subsequently:

- HasDrillUpButton
- HasGotoPageButton

Chapter 5

- HasPageNavigationButtons
- HasRefreshButton
- HasSearchButton
- HasZoomFactorList

So a typical use of these properties is where you want to give users a preview of the report with the ability to refresh the data shown. You could easily set a few properties before you set your ReportSource property to make this happen:

```
CrystalReportViewer1.HasRefreshButton = true
```

When the report is previewed, it will appear, as shown in Figure 5-18.

The screenshot shows a Microsoft Internet Explorer window titled "WebForm1 - Microsoft Internet Explorer". The address bar displays "http://localhost/web_viewer_properties/WebForm1.aspx". The page content is a "Product Inventory Report" for the "Competition" product type. On the left, there is a navigation tree with categories: Competition, Gloves, Helmets, Hybrid, Kids, Locks, Mountain, and Saddles. The main area displays a table with columns: PRODUCT NAME, COLOR, and SIZE. The table contains the following data:

PRODUCT NAME	COLOR	SIZE
Endorphin	deep	18.5
Descent	steel	15
Endorphin	deep	20
Endorphin	deep	17
Endorphin	deep	15
Mozzie	jewel	22
Mozzie	jewel	20
Mozzie	jewel	17
Mozzie	jewel	17

Figure 5-18

In addition to simple Boolean properties, there are also a couple of other properties that can be set to control the appearance and behavior of the viewer, as seen in the following table:

Report Integration for Web-Based Applications

Property	Description
PageToTreeRatio	For setting the ratio between the Group Tree and the rest of the page; larger numbers mean a larger report page, with a smaller Group Tree
PageZoomFactor	The initial zoom factor for the report when viewed

So if we wanted to change the `PageToTreeRatio` and zoom factor so that the report was presented a little bit better on the page, we could add the following code to be evaluated when the page was loaded:

```
CrystalReportViewer1.PageToTreeRatio = 7  
CrystalReportViewer1.PageZoomFactor = 80
```

Our previewed Web Form would look like Figure 5-19.

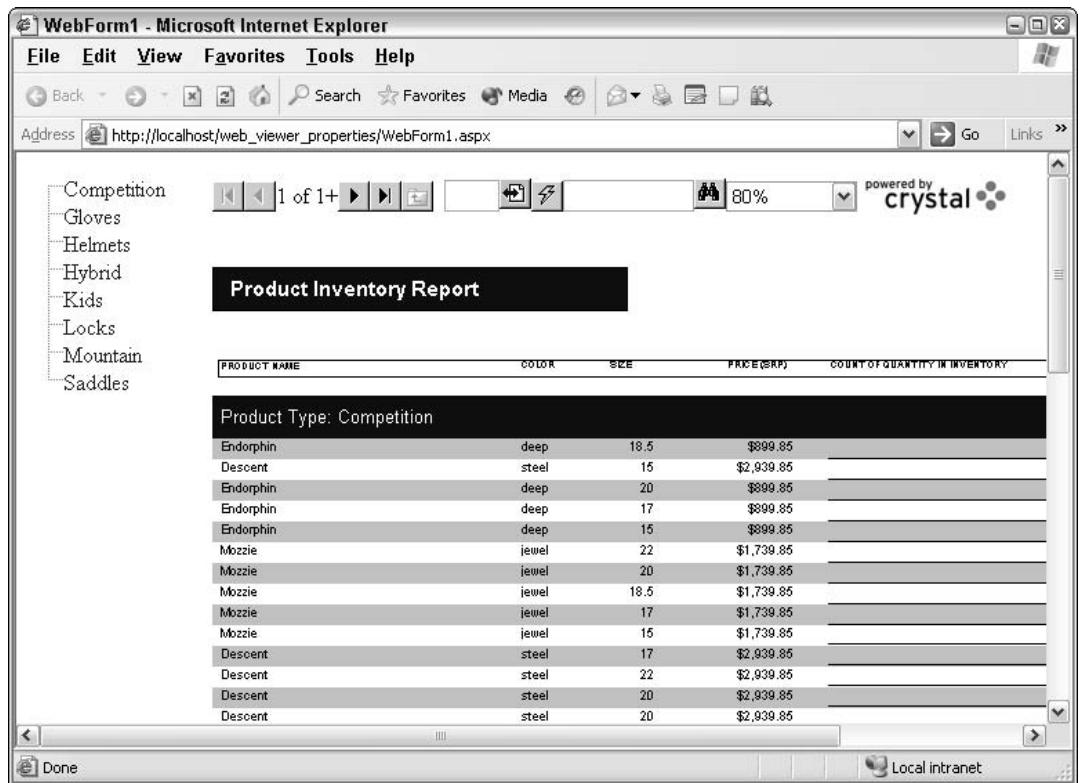


Figure 5-19

Viewer Methods

When working with the `CrystalReportViewer`, we have a number of methods available to us, which will allow us to integrate specific viewer functions into our application. As we move through this section,

Chapter 5

keep in mind that these methods can be used to create your own look and feel for the report preview window.

Create a new Web Form, which we'll call `web_viewer_methods`. Again, the code for this application is included with the download code. Drag a `CrystalReportViewer` onto this form. Include the report `product_listing_bytype.rpt` in your project (in the download code, the path is `Crystal.NET2003/Chapter05/product_listing_bytype.rpt`). Drag a `ReportDocument` component from the Toolbox onto your form, and when the dialog opens up, select `web_viewer_methods.product_listing_bytype` from the drop-down box. Click OK.

Now we add some code to tie our report to the application. In the `Page_Init` event in the designer generated code, once again add:

```
CrystalReportViewer1.DataBind()
```

Now all that remains is to set the `ReportSource` property in the `Page_Load` sub:

```
CrystalReportViewer1.ReportSource = cachedproduct_listing_bytype1
```

Compile and run this. Now, we're all set to customize our viewer.

In this example, we are actually going to walk through building a custom viewer. The first thing we need to do is set the `DisplayToolbar` property and the `DisplayGroupTree` property to `False` in the Properties pane for the viewer, and add some additional buttons and textboxes to our Web Form using the screen shot earlier as a guide, which we will walk through later in this article.

As we walk through this example, we are going to add the code behind these buttons and this form using the methods described later in this Chapter and learn how to match the viewer user interface to your own application.

Setting Browser Rendering

The `CrystalReportViewerBase` class provides a number of key properties, one of which is the `ClientTarget`. The `ClientTarget` property is a string and determines how the Crystal Report Viewer will render the report.

These strings are:

- `ie4`—for Internet Explorer 4.0
- `ie5`—for Internet Explorer 5.0
- `uplevel`—for most other Web browsers
- `downlevel`—for very basic Web browsers

A Web browser is considered `uplevel` if it can support the following minimum requirements:

- ECMAScript (JScript, JavaScript) version 1.2
- HTML version 4.0
- The Microsoft Document Object Model (MSDOM)
- Cascading style sheets (CSS)

Report Integration for Web-Based Applications

Browsers that fall into the `downlevel` category include those browsers that provide support only for HTML version 3.2.

So, to set the browser version you are targeting, you could set the `ClientTarget` property for your form like this, under the `Page_Load` subroutine:

```
CrystalReportViewer1.ClientTarget = "ie4"
```

There is also an `Auto` value, which is the default setting and automatically selects the best rendering option based on the browser type. Unless you are writing an application for a specific browser or compatibility level, leaving this property set to `Auto` will provide the best viewing experience for the browser you are using.

For more information on detecting the browser type your Web application is using, see the topic “Detecting Browser Types in Web Forms” in the Visual Studio .NET combined help collection.

Refreshing the Data in a Report

When a report is refreshed, it goes back to the database for the most current set of data available and runs the report again. On our custom Web viewer, you should have a Refresh button, so pull a Button control onto the Web Form and rename it `Refresh_Button` in the ID property in the Properties pane. Change the text property to `Refresh`.

Now, double-click the `Refresh_Button` on your form to open the code for it. We can add some code behind this button to refresh the report using the `RefreshReport` method as shown subsequently:

```
Private Sub Refresh_Button_Click(ByVal sender As System.Object, ByVal e  
    As System.EventArgs) Handles Refresh_Button.Click  
    CrystalReportViewer1.RefreshReport()  
End Sub
```

Compile and run the application. The button should now be present on your form. Click it. This will cause the report to return to the database and read the records again. Use this functionality with caution; if a report has a large SQL query to perform before it can return the first page, you may experience performance problems.

Page Navigation and Zoom

Now we are going to insert some buttons across the top of our Web Form in the same way we did with the Refresh button, with the following names and text values:

Button Name (ID Property Value)	Text Property Value
FirstPage_Button	First
Back_Button	Back
Forward_Button	Forward
LastPage_Button	Last

Chapter 5

We access these properties, once again, through the Properties pane in Visual Studio .NET.

For page navigation using the buttons we have drawn on our custom form, we have a number of methods that can be called without any additional parameters, as shown:

- ShowFirstPage
- ShowPreviousPage
- ShowNextPage
- ShowLastPage

These methods do not return a result, and unlike the Windows Forms Viewer, the Web Form Viewer does not have a `GetCurrentPageNumber` method, which would have returned an integer representing the page you are currently viewing.

To add these methods to the page navigation buttons, double-click the appropriate buttons on your Web Form and enter the code behind, as shown:

```
CrystalReportViewer1.ShowNextPage()
```

Do this for the other three buttons, including the appropriate method for each. Compile the project and test these buttons.

In addition to page navigation, you also have the ability to choose the zoom factor that is applied to your report. By default, the zoom is set to 100% of the report size unless you specify otherwise.

In our custom viewer, you should have a drop-down list for the zoom factor. To create our own zoom factor functionality, drag a drop-down list onto the form. Open the properties for your drop-down list (in our example, we have named the drop-down list `ZoomList`).

In the properties for your drop-down list, locate and open the `Items` property, which should open the dialog shown in Figure 5-20.

Using this dialog, we are going to create the items that will appear in our drop-down list and specify the corresponding values that will be passed to the form when an item is selected. Use the Add button to add items and make sure that the values correspond to the text you have entered (for instance, Full Size = 100, 50% = 50, and so on).

Once you have entered all of the values, click OK to accept these changes and return to your form's design. To use the `Zoom` method, double-click your drop-down box and add the following code:

```
CrystalReportViewer1.Zoom(ZoomList1.SelectedItem.Value)
```

This is simply calling the `Zoom` method using the item the user selects from your drop-down box. When you run your application and preview your custom viewer, you should be able to select your own zoom factor, and have it appear in the browser by pressing the Refresh button, as shown in Figure 5-21.

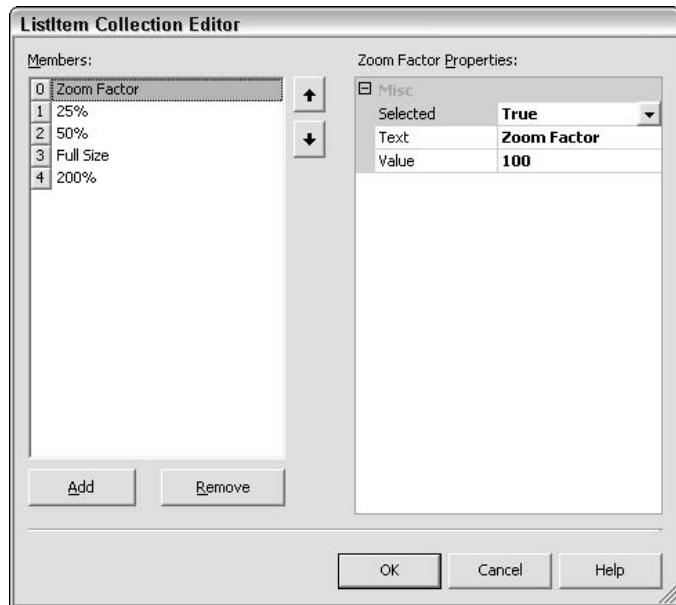


Figure 5-20

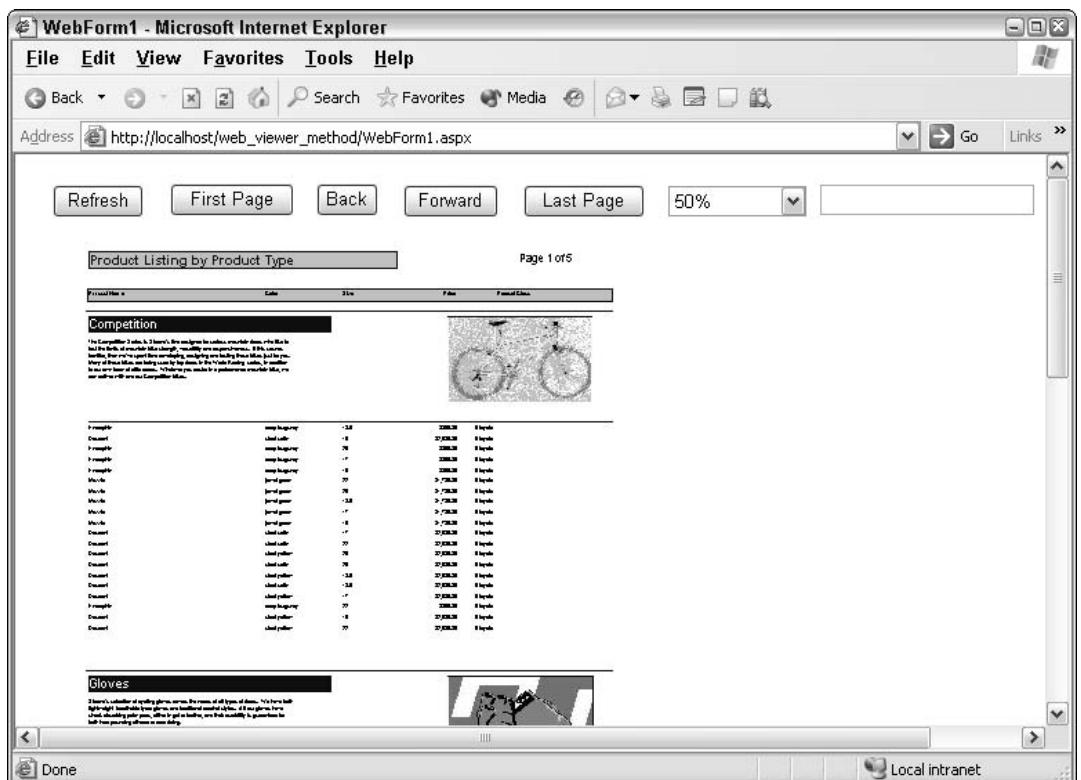


Figure 5-21

Searching Within a Report

Another powerful navigation feature can be found in the `SearchForText` method within Crystal Reports .NET, which will allow you to search for a specific string that appears in your report.

On our custom viewer, we are going to create a textbox and a button labeled Search. We are going to use this textbox to enter some search string and when the user clicks the Search button, we are going to use the `SearchForText` method to find the search string within our report.

To start, we will call our textbox `TextBox_SearchString` and our Search button `Search_Button`. Add these to the design view of our Web Form, remembering to replace the `Text` property for the button with `Search`.

To use the `SearchForText` method, double-click the Search button and add the following code behind:

```
Private Sub Search_Button_Click(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles Search_Button.Click  
    If TextBox_SearchString.Text <> "" Then  
        CrystalReportViewer1.SearchForText(TextBox_SearchString.Text, _  
            CrystalDecisions.[Shared].SearchDirection.Forward)  
    End If  
    TextBox_SearchString.Text = ""  
End Sub
```

The Crystal Report Viewer will search the entire report and when the value is found, go directly to the page on which it appears (the last line of the preceding code is just to clear the textbox for the next search). This method can be called repeatedly to find all of the occurrences of a particular string. Each time it finds the string in your report (in our example that follows, we searched for `Youth Helmet`), it will jump to that page, as shown in Figure 5-22.

You may have noticed that this method is slightly different between the Windows and Web viewers for Crystal Reports; with both types of Forms Viewer, you can pass the additional parameter of Search Direction for searching forward or backward through your report. However, this method in Windows will also highlight the found value. The Web version does not have this capability.

Printing Your Report

Now, if you have already done some report integration with Windows applications, you may have noticed that the Web Forms Viewer is missing one very important icon—the Print button. When a Crystal Report is viewed on the Web, it is actually rendered in static HTML 3.2 or HTML 4.0, with all of the report elements represented in HTML code and syntax.

This makes things difficult when it comes time to print your report. In a case where you were just using the plain old viewer with little or no modification, imagine if you were to click the print button from your browser to print your report.

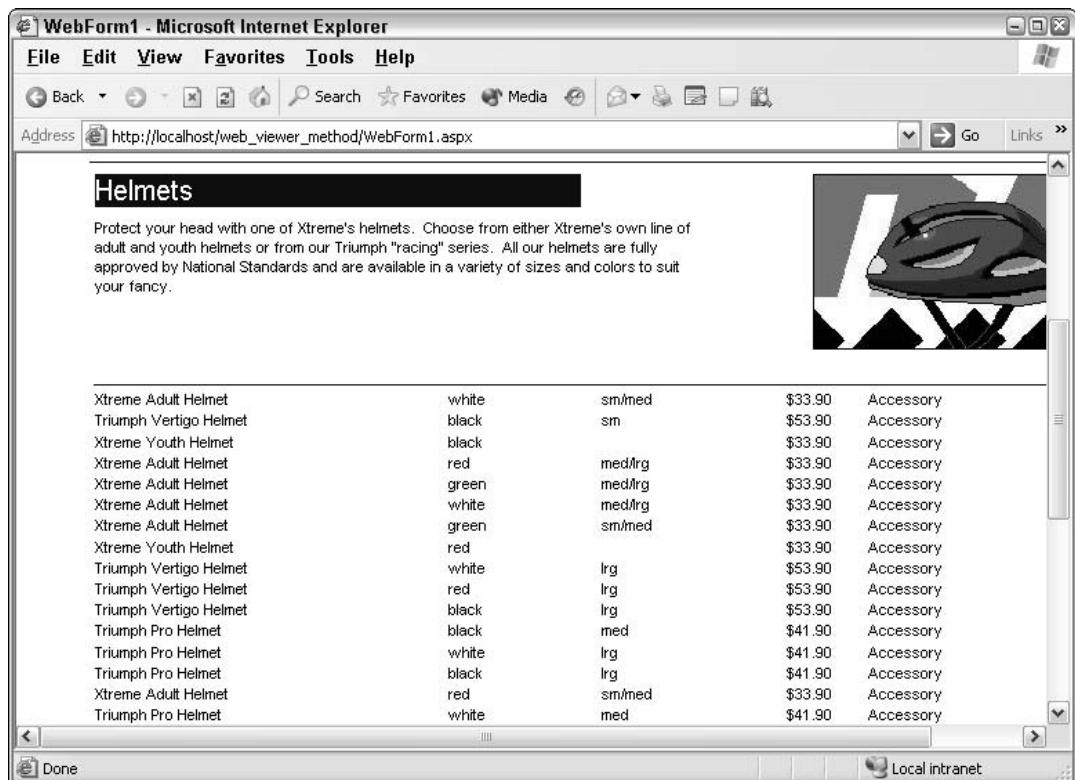


Figure 5-22

It is not going to be very pretty to say the least; if your application uses single-page reports or discrete parts of a report, you may be happy with this, but for the rest of us, there has to be a better solution. So in answer to this limitation in HTML and the way reports are presented in a browser window, we have to come up with some creative solutions to actually print our report to a printer.

The following sections detail the different ways a report can be printed from the Web, as well as some of the advantages and drawbacks of each method. Because we are looking at new functionality within Crystal Reports .NET, we are going to create a new project specifically for this section.

From within Visual Studio, select File → New → Project and from Visual Basic Projects, select ASP .NET Web Application and call this new application `web_viewer_print`. This application is included with the download code. To use the downloaded version a virtual directory should again be created for it in its downloaded location.

Once you have clicked OK, the development environment will open with the default form that we will be using in the section.

We also need to add a report to work with in this section, so select Project → Add Existing Item. Change the drop-down list to show All Files and specify `*.rpt` for the file name to filter the list to show only the

Chapter 5

available reports. The `web_printing_report.rpt` file is in the code download path `Crystal .NET2003\Chapter05\web_printing_report.rpt`.

Once you have selected the `web_printing_report.rpt` report, click Open and this report will be added to your project in the Solution Explorer. We will be looking at the different methods for printing this report in the following sections.

Now, simply drag a `ReportDocument` component onto the form, which should offer you `web_viewer_printing.web_printing_report` as first choice in the drop-down box. Select it and drag a `CrystalReportViewer` onto the Web Form. Now, to bind the `ReportDocument` component to the viewer, merely enter the following code in the Web Form's `Page_Init` event, as we have done more than once in this chapter:

```
CrystalReportViewer1.ReportSource = New web_printing_report()
```

Compile and run the application to check that everything is working. We are now ready to start looking at printing this report.

Printing from the Browser

The simplest method for printing a report is to print directly from the browser. You have already seen how well this works, but there are some tricks that we can use to improve the way the report prints if we are forced to use this method.

First, you can disable the `DisplayGroupTree` property if the report is likely to be printed. Do this by setting it to `False` in the Properties window, or you could do this programmatically by inserting the following code into the `Page_Load` event:

```
CrystalReportViewer1.DisplayGroupTree = False
```

The viewer object model provides a property called `SeparatePages` that by default is set to `True`, meaning that the report is chunked up into individual HTML pages based on the report pagination. When this property is set to `False`, the report itself becomes one long page, which can then be printed just like any other Web page. You can set this property through the property page of the Crystal Report Viewer, as shown in Figure 5-23.

Or you can also set this option programmatically:

```
CrystalReportViewer1.SeparatePages = False
```

Another trick is to actually turn off the toolbar and all of the icons so that the output on the page is close to what you would like to see when the report is printed.

```
CrystalReportViewer1.DisplayToolbar = False
```

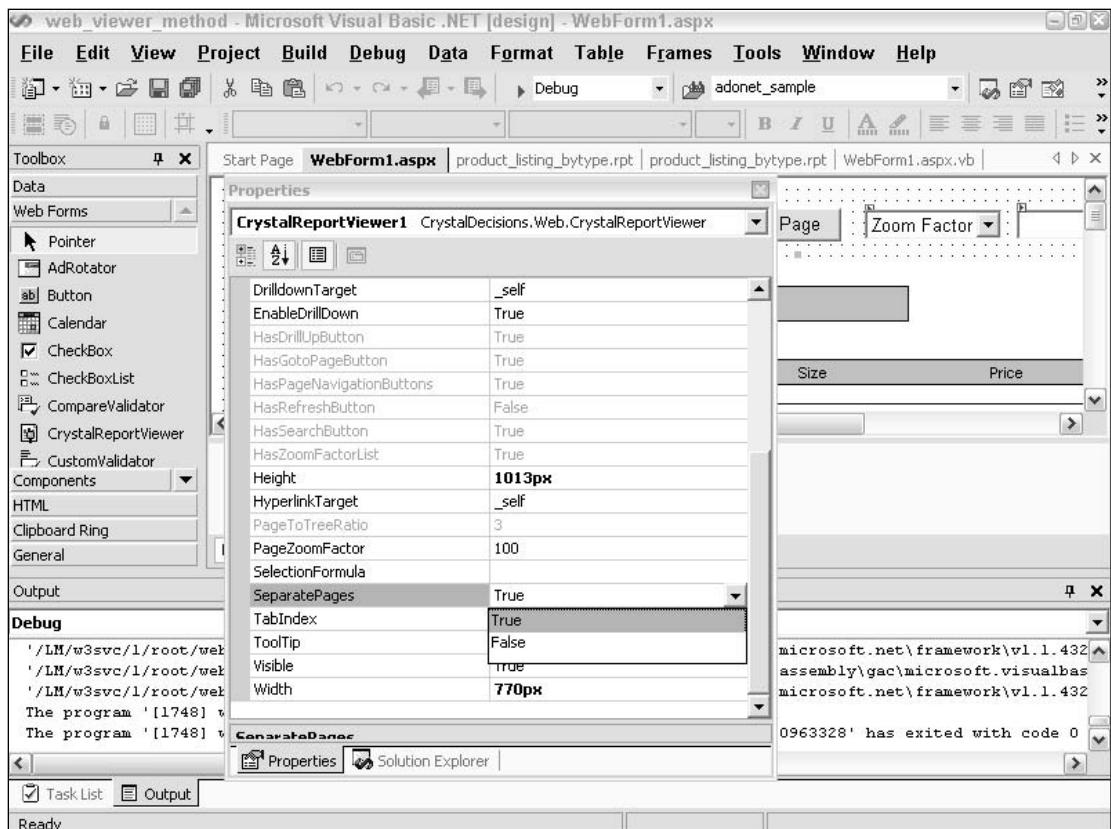


Figure 5-23

So with the toolbar turned off and our report showing as one long page, you can then print your report and have a somewhat decent output as shown in Figure 5-24, which is a preview from Internet Explorer.

The only problem is that this method does not take advantage of any of the neat formatting features for page headers and footers, as the browser just thinks this is one big page to be printed. In addition, the column headings are printed only on the first page, so it is difficult to read the report as you move through the pages.

This method is recommended only for reports with a small number of pages (1–20) as the entire report is concatenated into one long page, which may take a while to render on screen or print.

However, with that said, printing from the browser is the easiest method of printing your report from the Web, even with its limitations. For report developers who have put a lot of time and effort into their report design and want that report to be seen and printed by the users (and look good!), we need to look at another solution.

Chapter 5

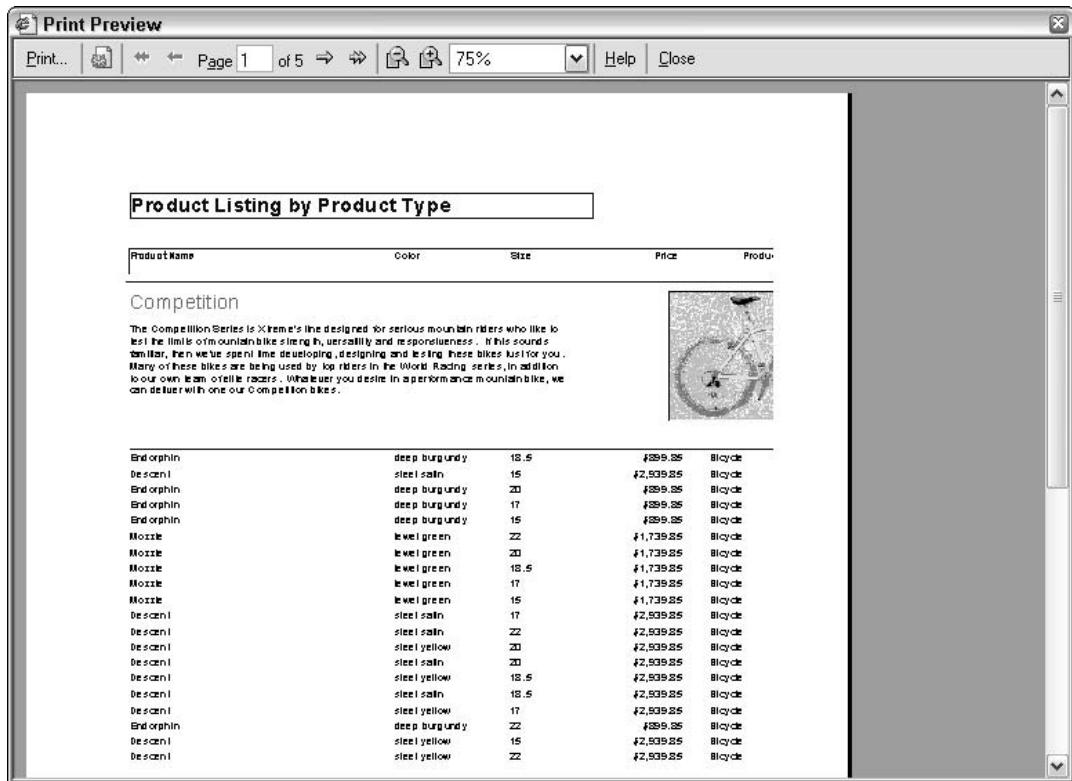


Figure 5-24

Printing from the Adobe Acrobat Plug-In

Crystal Reports .NET supports many export formats, and one of the more popular ones is Adobe's Portable Document Format, or PDF. Using the export functionality within Crystal Reports .NET and a copy of Adobe Acrobat Reader (or the plug-in) installed on the client machine, reports can be printed from the Web.

This is one of the methods recommended by Crystal Decisions for printing your reports in a presentation-quality format, and it actually developed the workaround used in this section to help developers who were used to the way Crystal Reports normally operates and were frustrated by not having that print button.

The first thing we need to do is create a new Web Form that will contain our instructions. We will call this form `AcrobatPrinter.aspx`, and create it by right-clicking the project name, and selecting Add → Add New Item. We will then select Web Form and name it as noted earlier. Right-click it and select Set As Start Page.

Draw or drag a button onto the Web Form and call it `PDF_Button`, and label it Export via PDF.

Report Integration for Web-Based Applications

Now we need to do some setup to utilize the Crystal Reports Engine (covered in Chapter 9, “Working with the Crystal Reports Engine”) and set some options available from the `CrystalDecisions.Shared` namespace.

So, we are going to put some code behind our export button to dimension variables for the export options that we want to use, and also for the specific options for exporting to a disk file. Double-click the button in the designer, and insert the following code:

```
Private Sub PDF_Button_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles PDF_Button.Click
    Dim myExportOptions As CrystalDecisions.Shared.ExportOptions
    Dim myDiskFileDestinationOptions As _
        CrystalDecisions.Shared.DiskFileDestinationOptions
```

Next, we are going to create a variable to hold the name of the file that we are going to be exporting to, as well as creating a new instance of a Sales Report that has already been added both to the project and to this form, through the `ReportDocument` component.

```
Dim myExportFile As String
Dim myReport As New web_printing_report()
```

For our next order of business, we need to set a temporary location for the output file (this can be anywhere on your server) and we are going to build a unique file name using the session ID from the ASP .NET session and tacking the PDF extension on the end, so the file association will work correctly when we go to view this file in our browser.

```
myExportFile = "C:\Crystal.NET2003\Chapter05\PDF " & _
    Session.SessionID.ToString & ".pdf"
```

Now, for the meat of the matter—actually setting the destination options to export your report to a PDF file and write it to the disk.

```
myDiskFileDestinationOptions = New
    CrystalDecisions.Shared.DiskFileDestinationOptions()
myDiskFileDestinationOptions.DiskFileName = myExportFile
myExportOptions = myReport.ExportOptions
    With myExportOptions
        .DestinationOptions = myDiskFileDestinationOptions
        .ExportDestinationType = .ExportDestinationType.DiskFile
        .ExportFormatType = .ExportFormatType.PortableDocFormat
    End With
```

Then we call the `Export` method to export our report:

```
myReport.Export()
```

But we are not done yet! We need to take the exported PDF file that has been generated and output it to the browser so the user can view it using the Acrobat Plug-In or standalone viewer. To do that, we are going to use some simple response statements to return the file to the browser:

Chapter 5

```
Response.ClearContent()
Response.ClearHeaders()
Response.ContentType = "application/pdf"
Response.WriteFile(myExportFile)
Response.Flush()
Response.Close()
```

Finally, once we have delivered the file to the user, we need to clean up after ourselves and remove the file from the server altogether.

```
System.IO.File.Delete(myExportFile)
End Sub
```

So when all of this code is put together behind our export button and our application is run, the user can click the button and preview and print the report from Adobe Acrobat, with the page numbering and other features in place.

Printing from Other Export Formats

In addition to Adobe Acrobat format, you can also print to other supported export formats such as Excel, Word, or others, by changing the file extension, the MIME type, and the `ExportFormatType` property in the aforementioned code. There are a number of different destinations that are supported, including:

Name	Description	MIME Type
Excel	To export to a Microsoft Excel file	Application/vnd.ms-excel
HTML32	To export to an HTML file compatible with HTML v3.2	Application/html
HTML40	To export to an HTML file compatible with HTML v4.0	Application/html
PortableDocFormat	To export to PDF (Acrobat) format	Application/pdf
RichText	To export to an RTF file for use with Microsoft Word, WordPerfect, and so on	Application/rtf
WordForWindows	To export to a Microsoft Word file	Application/msword

If you want to export to Word, the RTF export actually provides a better export format. To open the RTF on the client side using Word (instead of the application associated with the RTF file extension), leave the `ExportFormatType` property set to `RichText`, but change the MIME type to be `application/msword`.

Using Viewer Events

Viewer events provide the ability to track when different events are fired from the browser—for instance, when the user navigates through the pages of the report or when they refresh the report. These events can then be used to fire other code within your application.

Report Integration for Web-Based Applications

Although all of the different events have their own unique properties and methods, they all inherit a common property called `Handled` that is a Boolean value used to determine whether the event was fired and subsequently handled.

In the following section, we will be looking at all of the available events associated with the viewer and their common use. If you would like to try out some of the following events, open the custom viewer we were working with earlier in the chapter (`WebForm1.aspx` from the project `web_viewer_properties`) and add a label to your form (call it `Event_Label`); we'll use this label to notify the user when an event is fired. Clear its `Text` property. Now we are ready to begin.

Page Navigation Events

For page navigation, the `NavigateEventArgs` class provides the properties we need to work with the `Navigate` event, including:

Property	Description
<code>CurrentPageNumber</code>	Returns the current page number
<code>NewPageNumber</code>	Gets or sets the new page number

In the following example, the `Navigate` event would fire when a user changed the page within the viewer, resulting in a label that would show the page they are coming from and the page they are navigating to.

Insert the following subroutine into your Web Form code:

```
Private Sub CrystalReportViewer1_Navigate(ByVal source As Object, ByVal
    MyEvent As CrystalDecisions.Web.NavigateEventArgs) Handles
    CrystalReportViewer1.Navigate

    If MyEvent.NewPageNumber <> 1 Then
        Event_Label.Text = "Current page: " & MyEvent.CurrentPageNumber &
            " New Page: " & MyEvent.NewPageNumber
    End If
End Sub
```

So, as the user navigates through the pages, this information is shown and can be used in your application. Compile and run this code to see this happen. (Make sure you set `WebForm1` as the startup form for your project if it isn't already.)

Refresh Events

The `ReportRefresh` event has no arguments other than the inherited `Handled` property. It can be used to build metrics on how often a report is run or refreshed, and to pass information to users about the report before they launch a refresh, as shown:

Chapter 5

```
Private Sub CrystalReportViewer1_ReportRefresh(ByVal source As Object,
    ByVal MyEvent As CrystalDecisions.Web.ViewerEventArgs) Handles
    CrystalReportViewer1.ReportRefresh
    Event_Label.Text = "Please be advised this report takes up to 2 minutes
        to run."
End Sub
```

Insert this subroutine into your Web Form code, in the same way we did earlier. Compile and run. The message should now appear in the label when you hit Refresh.

Search Events

When a user searches for a report value, either through the standard icon on the toolbar or through your own method call, the Search event is fired. The arguments for the Search event are:

Property	Description
Direction	Gets or sets the direction in which to search. This can be either Backward or Forward.
PageNumberToBeginSearch	Gets or sets the page number on which to start searching.
TextToSearch	Gets or sets the text to search for in the report.

So by using these event arguments, you could keep a record of what values users searched for or offer a Top 10 search facility to let them search using the 10 most requested search strings. An example of getting the text that is being used in the search follows. Insert this subroutine into your code, build and run it:

```
Private Sub CrystalReportViewer1_Search(ByVal source As Object, ByVal
    MyEvent As CrystalDecisions.Web.SearchEventArgs) Handles
    CrystalReportViewer1.Search
    Event_Label.Text = "You searched for " & MyEvent.TextToSearch
End Sub
```

Zoom Events

When the user changes the zoom factor for a particular report, the ViewZoom event fires, and has only one argument, ZoomEventArgs. The NewZoomFactor property will get or set the magnification factor for the viewer, as shown here:

```
Private Sub CrystalReportViewer1_ViewZoom(ByVal source As Object, ByVal
    MyEvent As CrystalDecisions.Web.ZoomEventArgs) Handles
    CrystalReportViewer1.ViewZoom
    Select Case MyEvent.NewZoomFactor
        Case "25"
            Event_Label.Text = "You have selected 25%"
        Case "50"
            Event_Label.Text = "You have selected 50%"
        Case "100"
            Event_Label.Text = "You have selected full size"
    End Select
End Sub
```

For further customization of your report and control of your report's features and functionality, you may want to turn to Chapter 9, "Working with the Crystal Reports Engine" to learn how to work with the Crystal Reports Engine, which provides control over your report at run time.

Summary

By now you know how to integrate reporting into both your Windows and Web applications, with this chapter focusing on the latter. You should be able to pick the right object model for the functionality you want to provide to your users, as well as work with all of the properties, methods, and events contained within those models.

For our next trick, we are going to look at extending Crystal Reports through the use of XML Report Web Services, which is the topic of Chapter 6, "Creating SML Report Web Services."

6

Creating XML Report Web Services

In the previous chapters, we had a look at how to integrate reports into Windows and Web-based applications, but now we need to learn how to leverage those skills and work with XML Report Web Services. First introduced with Visual Studio .NET 2002, XML Report Web Services provide a way to share reports and information with users both in and outside of your organization.

In this chapter, we will be looking at:

- An XML Report Web Services overview
- Creating XML Report Web Services
- Consuming XML Report Web Services
- Deploying applications that use XML Report Web Services

At the end of this chapter, you will be able to identify what an XML Report Web Service is and understand how it can be used in your application. You should also be able to create a Report Service from an existing Crystal Report and utilize the service with the Crystal Windows or Web viewer to view the report.

Obtaining the Sample Files

All the example reports and code used in this chapter are available for download. The download file can be obtained from www.wrox.com.

There is a compiled Web service that is included in the download files, including all of the examples shown throughout the chapter. However, it is simpler to create Web services as opposed to

setting up precompiled code, though an installer is included with the download code. Therefore, we recommend that you build your own according to the instructions in this chapter.

XML Report Web Services Overview

In the past, there have been a number of different methods for sharing information between different companies or organizations. Over the years, a number of standards and standard file formats have emerged, but each had its own unique strengths and weaknesses. EDI, for instance, was created to exchange data between companies (usually for purchasing or other supply-chain related use), but the EDI format was exacting and cumbersome for developers to use in their applications.

If you have worked as a developer for long, chances are you have made your own ad hoc attempt at information exchange—through extracts, data interchange, database replication, and synchronization. Although these methods may provide information to the people who need it, it is very time-consuming trying to impose some standards on these processes and the information itself.

Even when information was successfully shared between organizations, it was then a question of “What do we do with it now?” Often, another database instance would have to be created, transformation and loading routines developed, and finally, another suite of reports would have to be created, resulting in duplication of effort between whoever owned the data and the organization they were sharing it with.

With the introduction of XML Web Services with Visual Studio .NET 2002, and in particular, XML Report Web Services provided by Crystal Reports .NET, some of these information sharing and integration problems can be solved.

In its most simple terms, an XML Web Service is a piece of code that provides a specific function (or set of functions) that can be shared between different development environments and applications. The fact that these services are based on common standards, like XML, SOAP, and HTML means that this technology can be leveraged across a number of different applications or uses.

Using an XML Web Service, you could encapsulate a snippet of code to process credit card transactions, for example, and compile this code to a Web service. From that point, any number of different applications on different platforms, local and remote, could use that same Web service and the functionality it provides.

The same concept applies to reports you may have developed. Through the use of XML Report Web Services, a developer can create a feature-rich report that can be compiled and used (and reused) by information consumers and application developers without losing any of the inherent formatting or features.

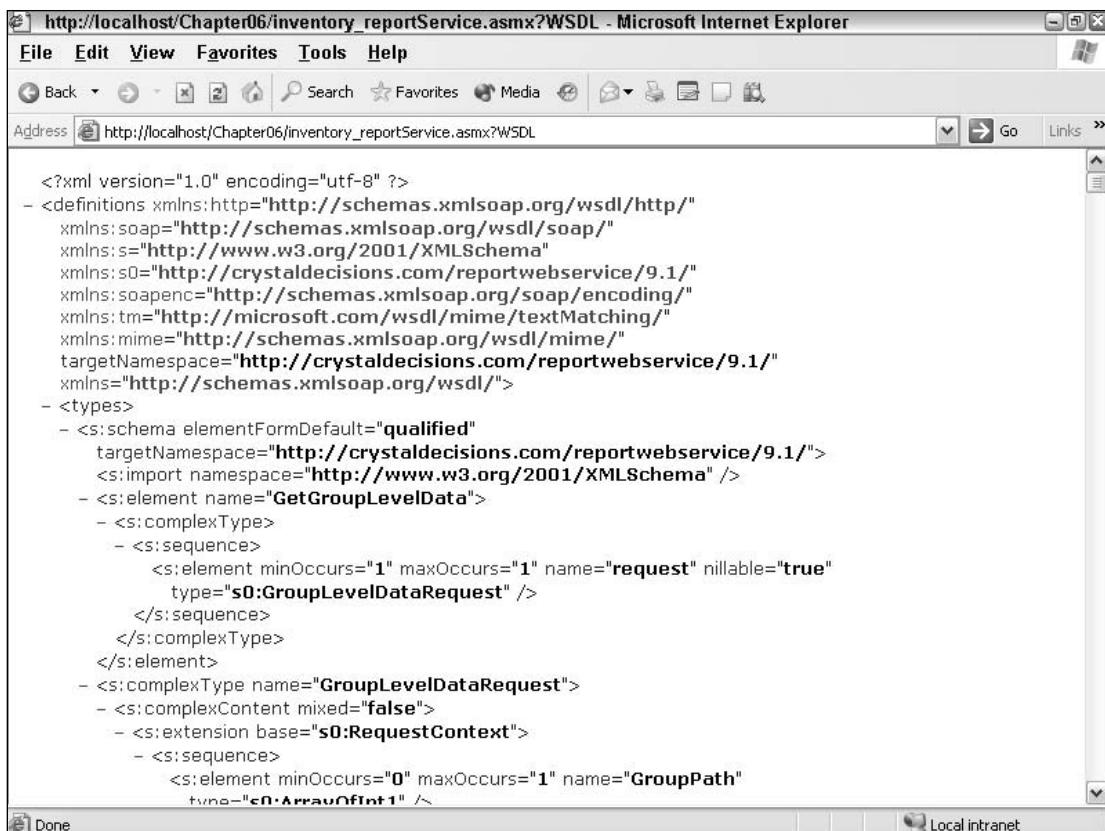
What Are XML Report Web Services?

The simplest definition of an XML Report Web Service is that it is a report file that has been published as a Web service, which can be consumed and viewed on any number of different platforms (including Windows and Web-based applications created with Visual Studio .NET or other tools).

Creating XML Report Web Services

During the publishing process, Visual Studio examines your report file, its content, the layout of the report, and which features are used when creating a Report Web Service from your report file. Visual Studio will create a .dll file and then take all of the attributes, including the data types for the fields you have selected, parameters, and resulting data types, and generate an XML file to describe these attributes.

When the Report Web Service is used or consumed, another special type of XML file is created, called a *Web Service Description Language* (.wsdl) file, as shown in Figure 6-1.



The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost/Chapter06/inventory_reportService.asmx?WSDL - Microsoft Internet Explorer". The address bar contains the URL "http://localhost/Chapter06/inventory_reportService.asmx?WSDL". The main content area displays the WSDL XML code. The XML defines a service with various operations, including "GetGroupLevelData" and "GroupLevelDataRequest", with their respective request and response structures. The XML uses namespaces such as http://schemas.xmlsoap.org/wsdl/http/, http://schemas.xmlsoap.org/wsdl/soap/, and http://www.w3.org/2001/XMLSchema, and includes specific types like "qualified" and "true".

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:ns0="http://schemas.xmlsoap.org/wsdl/" xmlns:ns1="http://www.w3.org/2001/XMLSchema" xmlns:ns2="http://crystaldecisions.com/reportwebservice/9.1/" xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns4="http://microsoft.com/wsdl/mime/textMatching/" xmlns:ns5="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://crystaldecisions.com/reportwebservice/9.1/" xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
- <s:schema elementFormDefault="qualified"
  targetNamespace="http://crystaldecisions.com/reportwebservice/9.1/">
  <s:import namespace="http://www.w3.org/2001/XMLSchema" />
- <s:element name="GetGroupLevelData">
  - <s:complexType>
    - <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="request" nillable="true"
        type="ns0:GroupLevelDataRequest" />
    </s:sequence>
  </s:complexType>
</s:element>
- <s:complexType name="GroupLevelDataRequest">
  - <s:complexContent mixed="false">
    - <s:extension base="ns0:RequestContext">
      - <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="GroupPath"
          type="ns0:ArrayOfInt" />
```

Figure 6-1

This particular Web service ships with the sample code for this chapter and is built from the inventory report (`inventory_report.rpt`), also included in the sample files.

This file is written using Web Services Description Language, or WSDL. For applications and users that will access this Report Web Service, this file documents how they can interact with the service itself.

Chapter 6

Normally, if you were an application developer creating a Web service from scratch, you would develop most of these components yourself using the tools available within Visual Studio .NET and the .NET Framework. With XML Report Web Services, Crystal Reports does most of the work for you.

If you are interested in creating other types of Web services from scratch, find a copy of Professional ASP .NET 1.0 and check out Chapter 19, "Exposing Web Services." This chapter also provides some good background information on Web services and the different protocols used with them.

How Would I Use an XML Report Web Service?

The most common scenario for using an XML Report Web Service is when you need to share a report or its content with another department or organization. One example of when this type of Web service would be useful is managing the supply chain and inventory between a vendor and its customer.

To keep customers in the loop, a vendor could create a number of Crystal Reports that display all of the required information for backorders, shipped orders, and item availability. The vendor could then create XML Report Web Services from these reports and advertise the availability of these Web services to their customers.

For larger customers who already have an intranet or other vehicle for displaying the content from these Web services, their application developers could create a few simple pages to view the reports served up by the Web services. Instead of having a report sent to them or viewing a report snapshot, they would actually have access to live data in their report, served directly from the vendor's data.

For smaller customers who don't have developers who can provide an interface to these Web services, the vendor may choose to create its own application that gives these types of customers access to the reports available in the Web services.

In either scenario, there is tremendous value to both the vendor and its customer; information is provided in real time, with no additional effort required to update a data mart, or produce and send reports or extracts. All of the manual effort required to deliver this type of solution in the past is no longer required.

In the following sections, we are going to walk through creating XML Report Web Services and applications that can consume them.

Creating XML Report Web Services

The process of creating XML Report Web Services is relatively simple and features a number of shortcuts to help cut down on development time. Unlike the process you would use to create Web services from scratch, there is no custom coding required to publish an existing Crystal Report file to a XML Report Web Service.

Creating Basic Report Web Services

To get started, we need to create a new Windows application using Visual Basic .NET. If you want to follow along with the sample code that is available for this chapter, you can open the solutions file (Chapter6.sln) where you will find a number of projects that correspond to the sections in this chapter. To create your own project as we go along, create a new project from within Visual Studio by selecting File → New → Project and from Visual Basic Projects, select ASP .NET Web Service and specify a name (in the sample code, we have called this project WebService1) and location on your Web server for your project files (see Figure 6-2).

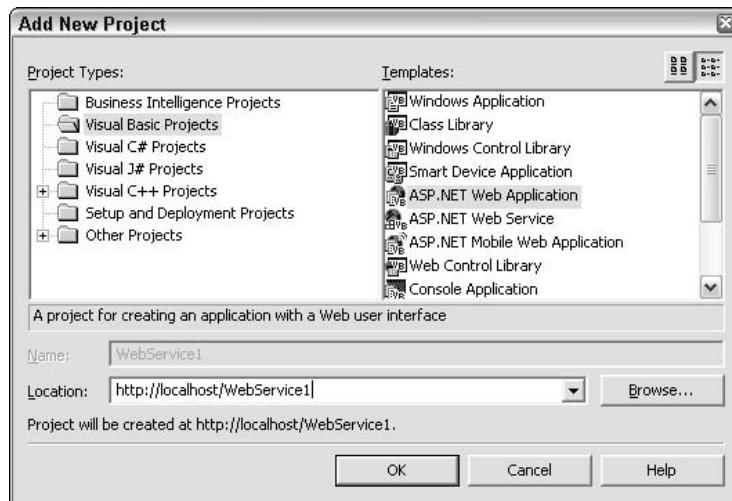


Figure 6-2

In this example, we are going to create a Report Web Service from the inventory report (`inventory_report.rpt`), one of the reports included with the code samples (in our installation we have downloaded this to `C:\Crystal.NET2003\Chapter06\inventory_report.rpt`; you may wish to alter this depending on your own installation). To add this report to your Web service, select Project → Add Existing Item, change the file extension to `.RPT`, and browse to where you have unzipped your code files and select the `inventory_report.rpt`.

Once you have added this report to your project, it should appear in the resources, and you can edit the report design and content as required. For now, we'll leave the content as is until we start looking at some of the more advanced features.

With the report added, simply right-click the report itself in the Project Explorer and from the right-click menu, select Publish as Web Service, as shown in Figure 6-3.

Chapter 6

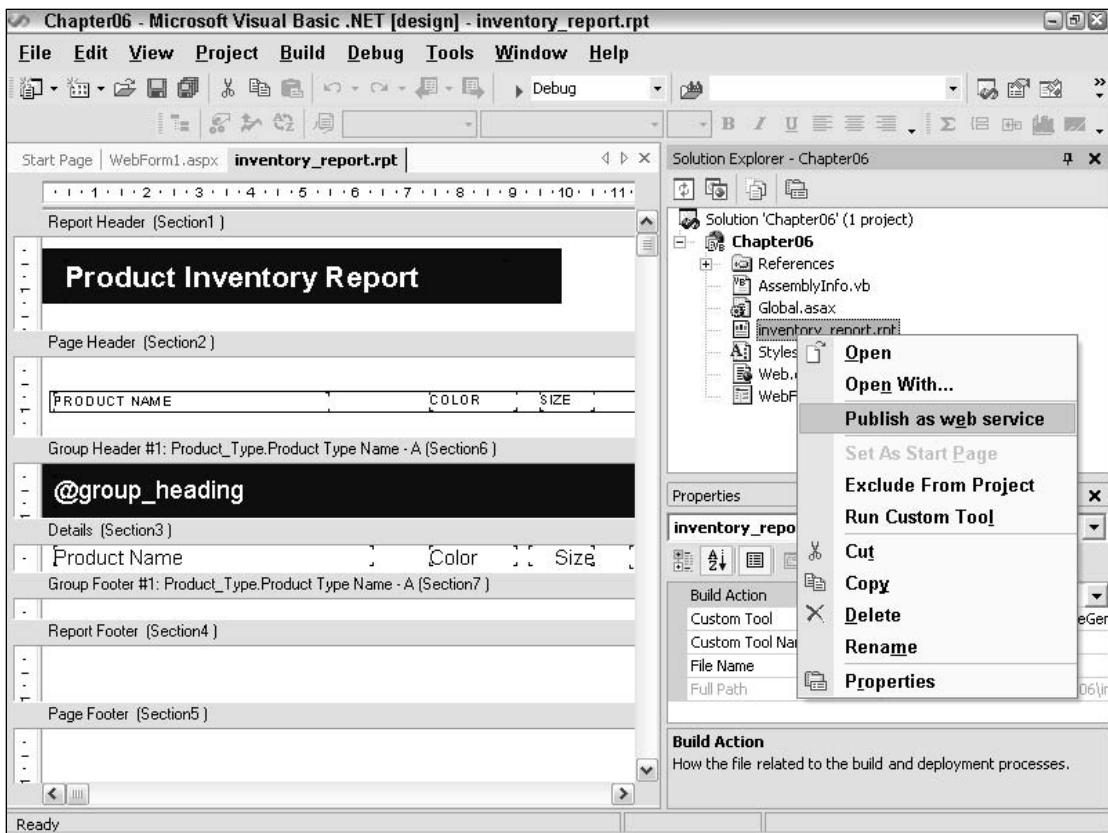


Figure 6-3

This will in turn generate an ASMX file that will also appear in the Project Explorer. There is also a source file generated for your Report Web Service, but by default, it is hidden. To show the source file, select the option for Show All Files in the Solution Explorer and you should see all three files, as shown in Figure 6-4.

Remember to compile `inventory_reportService.asmx` before you attempt to run it. You can do this by selecting Build → Build Solution or by hitting F5 on your keyboard.

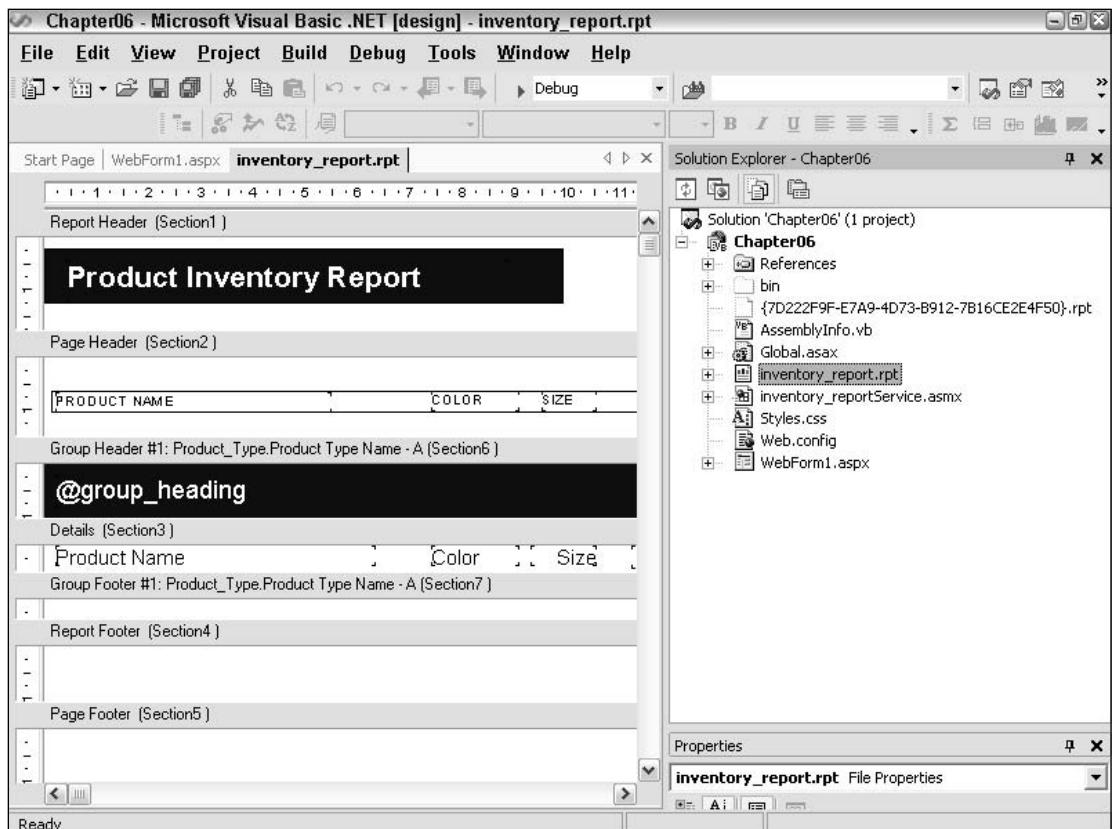


Figure 6-4

For Report Web Services created using Visual Basic, the Web service file will have the extension .vb attached on the end. You can alter this source file to add additional functionality, launch other Web services, and perform other functions.

With your report published as a Web service, you should be able to right-click directly on the corresponding .asmx file and select View in Browser from the right-click menu to display a list of valid operations for your Web service, as shown in Figure 6-5.

Chapter 6

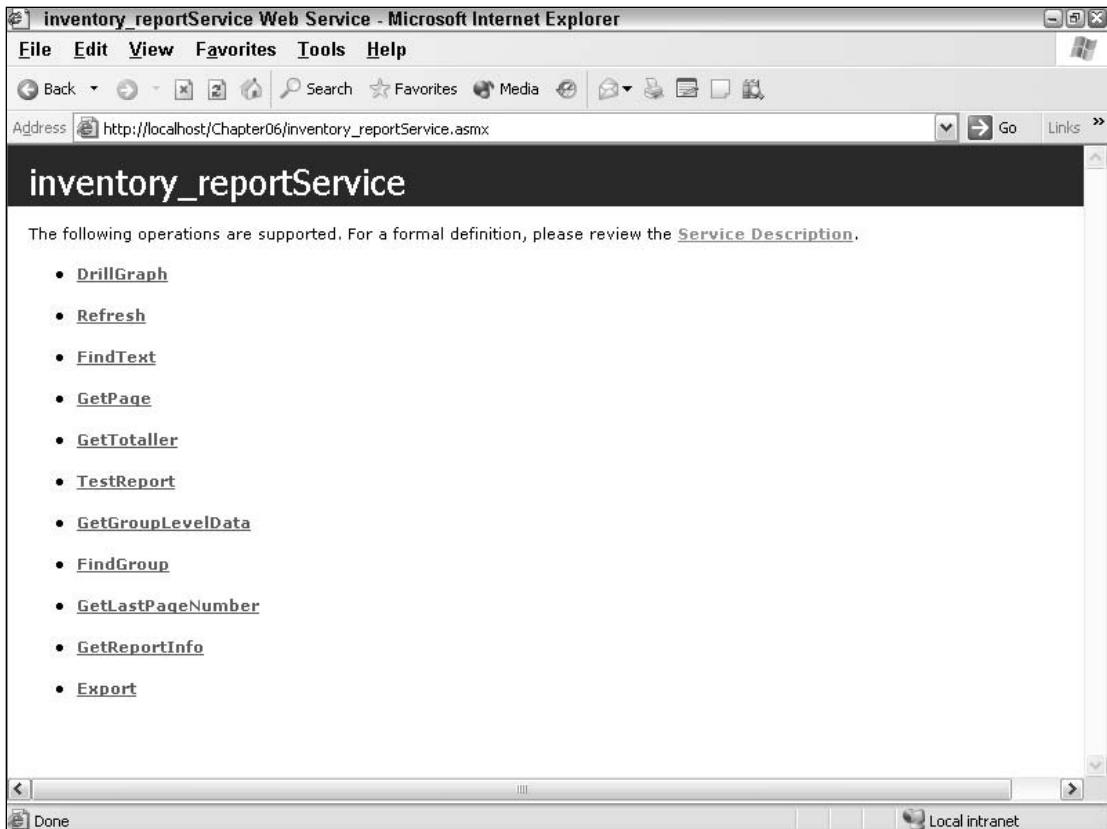


Figure 6-5

The URL for your Web service in this example is:

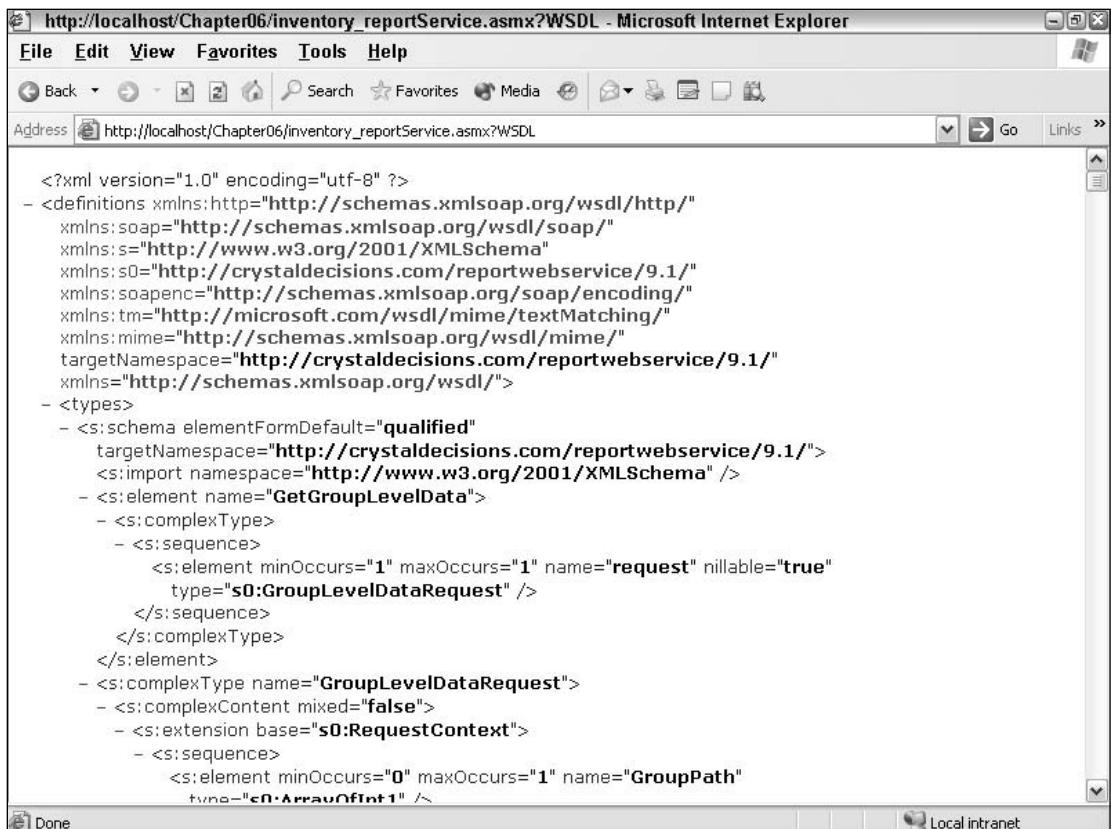
```
http://localhost/Chapter06/inventory_reportService.asmx
```

However, if our report name had a space in it, for example “Inventory Report,” the space would be encoded and the URL would look like this:

```
http://localhost/WebService1/Inventory%20ReportService.asmx
```

A good practice for Report Web Services is to ensure that the name of your report does not have any spaces in it; in this instance, renaming the report to remove the space or replacing it with an underscore would probably be easier than remembering to put in the spaces or correct encoding when calling your Report Web Service.

To toggle the formal definition, select the link for Service Description to display the XML document shown in Figure 6-6.



The screenshot shows the Microsoft Internet Explorer browser window displaying the WSDL (Web Services Description Language) for the service `http://localhost/Chapter06/inventory_reportService.asmx?WSDL`. The WSDL code is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:ns0="http://schemas.xmlsoap.org/wsdl/">
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:s0="http://crystaldecisions.com/reportWebService/9.1/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tme="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://crystaldecisions.com/reportWebService/9.1/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
  - <s:schema elementFormDefault="qualified">
    targetNamespace="http://crystaldecisions.com/reportWebService/9.1/"
    <s:import namespace="http://www.w3.org/2001/XMLSchema" />
  - <s:element name="GetGroupLevelData">
    - <s:complexType>
      - <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="request" nillable="true" type="s0:GroupLevelDataRequest" />
      </s:sequence>
    </s:complexType>
  </s:element>
  - <s:complexType name="GroupLevelDataRequest">
    - <s:complexContent mixed="false" />
      - <s:extension base="s0:RequestContext" />
        - <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GroupPath" type="c0:ArrayOfInt" />
        </s:sequence>
      </s:extension>
    </s:complexContent>
  </s:complexType>
</s:schema>
</types>
</definitions>
```

Figure 6-6

When we look at consuming XML Report Web Services a little later in this chapter, we will look at some of these methods and their use.

Creating Report Web Services with Multiple Reports

If you have multiple reports that you would like to publish as XML Report Web Services, you can place all of these report files into a Web service project by following the same procedure as a single report file.

Simply add each report from the Project → Add Existing Item menu and then right-click each and select Publish as Web Service. To view the Report Web Service for a specific report, just reference the name of the correct .asmx file associated with that report. Again, remember to compile them before trying to view them.

For example, if you were to add a report named `sales_graph.rpt` to the project we have been working with and publish it as a Web service, the URL reference would be:

```
http://localhost/Chapter06/sales_graphService.asmx
```

Chapter 6

This is an easy way to keep related reports and services together and can serve as the basis for a naming and hierarchy structure for your Report Web Services.

Utilizing the Generic Report Web Service

In addition to creating individual Report Web Services for each report you wish to publish as a Web service, Crystal Reports .NET also includes a generic Web Report Service, which supports what are known as Server Reports.

Server Reports are Crystal Report files that can be accessed through a generic Report Web Service (`ServerFileReportService.asmx`) and are available for use from the Server Explorer within the Visual Studio .NET IDE, as shown in Figure 6-7.

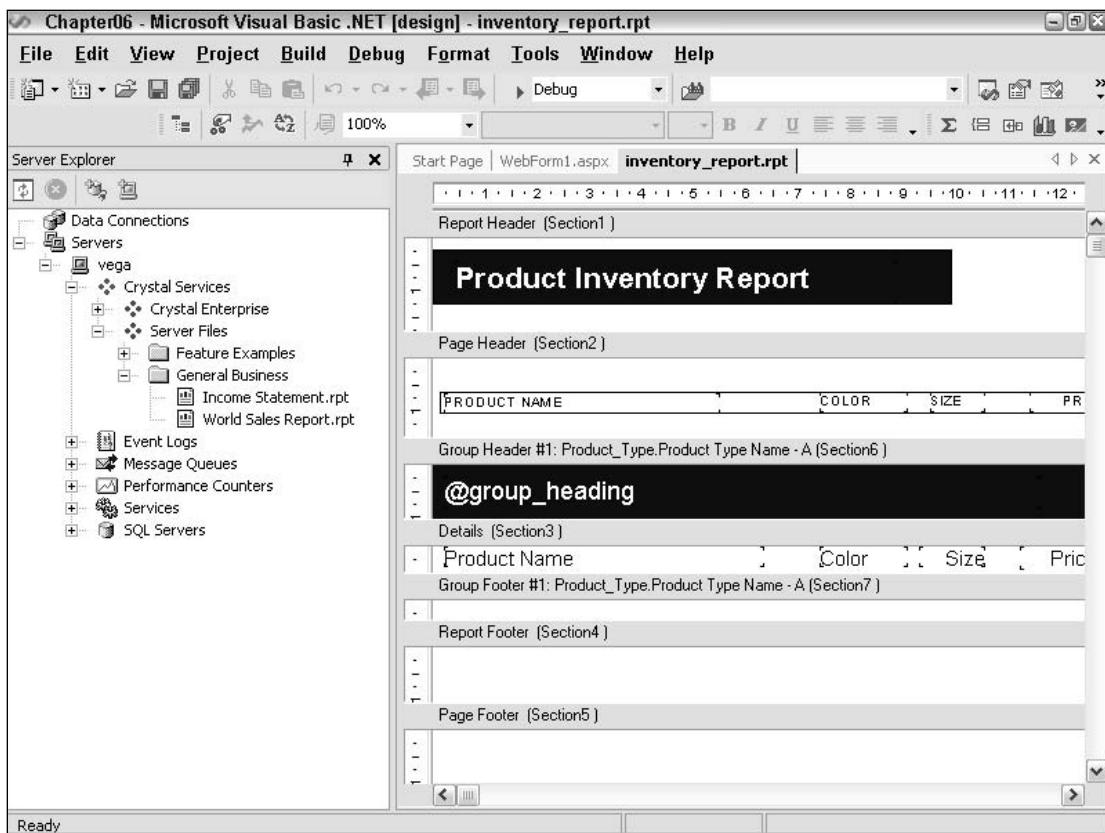


Figure 6-7

When working with the generic Report Web Service, there is no need to publish each report individually, but there are a couple of properties that you will need to set when viewing your reports using this method.

To view a report through the generic Report Web Service, you will need to specify the Web service URL that points to the generic Report Web Service, which, in turn, accesses the report specified in the ReportPath property. With the ReportPath, you don't actually need to put in the full path of the report because this property references a root directory that is embedded into the application.

This file also controls what reports you will see in the IDE under the Server Files node of the Crystal Services branch of your server.

When you first install Crystal Reports .NET, the RootDirectory tag within the file refers to the sample reports that ship with the product, so when you browse the Server Files node of the Server Explorer within the Visual Studio IDE, you are actually starting at the C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Reports directory. The URL to access this Web service is:

```
http://localhost/crystalreportviewer10/ServerFileReportService.asmx
```

If you don't see any reports underneath the Server Files node, check to see if you have the Web services component of Crystal Reports .NET installed. If you took all of the defaults on the Visual Studio .NET installation, you wouldn't have this option installed, as shown in Figure 6-8.

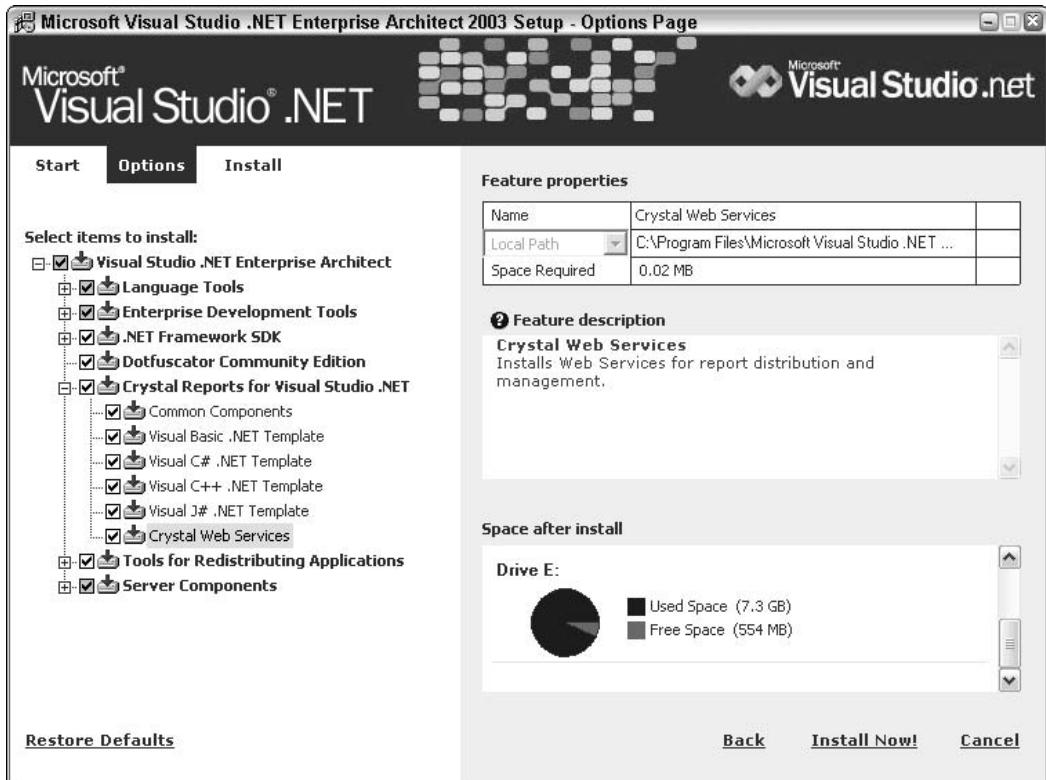


Figure 6-8

You will need to rerun the setup wizard to add this component before you can use the generic Report Web Service. To check that the services have been installed correctly, check out the directory `c:\program files\Microsoft Visual Studio .NET 2003\Crystal Reports\Viewers` to make sure that the file `ServerFileReportManager.asmx` is present.

Consuming XML Report Web Services

When consuming XML Report Web Services, the `.asmx` file associated with the service provides the entry point for consumption. When you request this `.asmx` file without any additional parameters or strings attached, the service will display a default help page like the one shown in Figure 6-9, listing all of the methods that are available for the service.

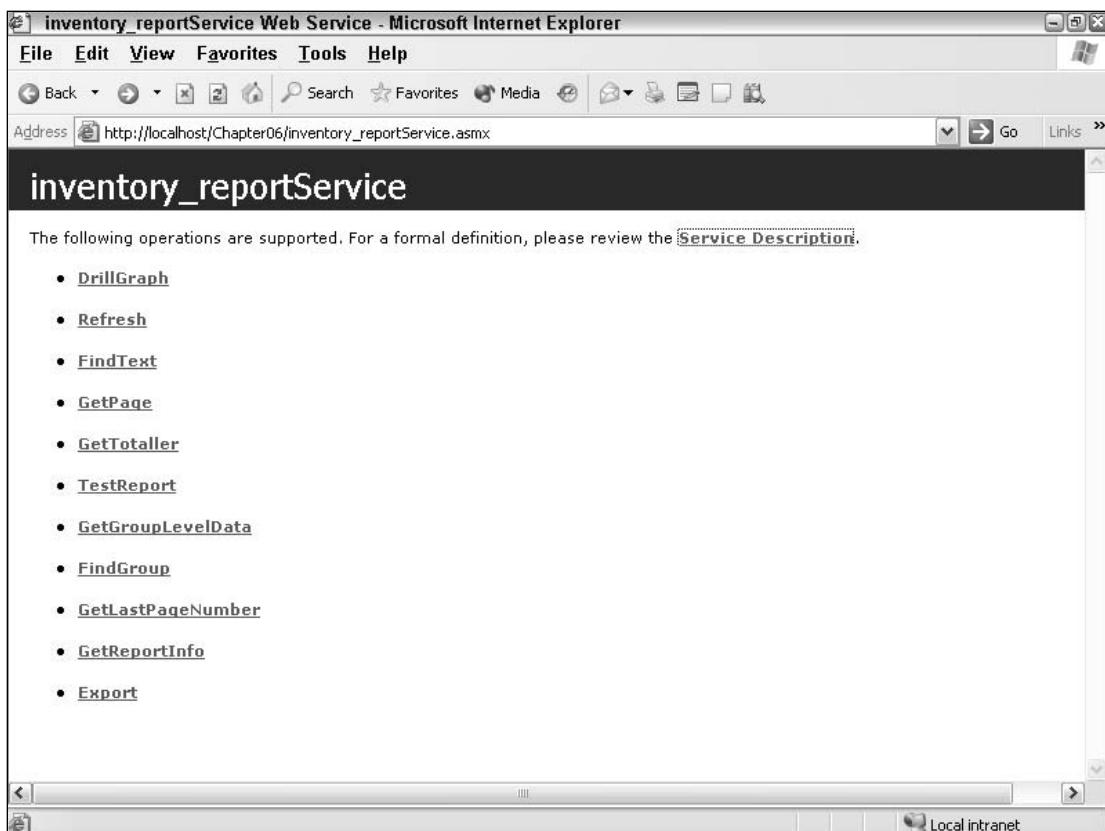


Figure 6-9

From the listing of methods, you can drill down for further information about their use. In addition, you can invoke methods that support using the HTTP-POST protocol. With Report Web Services, the only method that supports the POST protocol is the Test Report method shown in Figure 6-10.

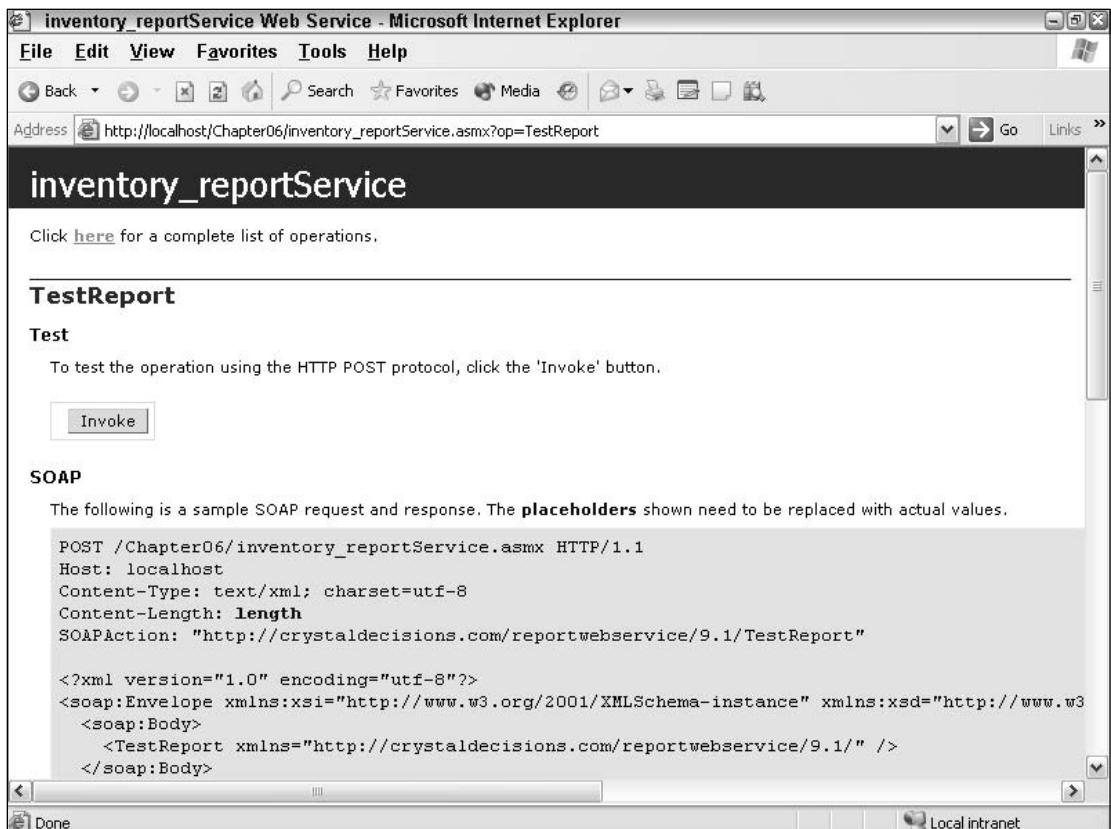


Figure 6-10

When you invoke this method, the result should be a summary of the report information, including the title of the report and the file name.

External Report Web Service

The first method we are going to look at for consuming an XML Report Web Service from an application uses a direct URL call to the Report Web Service from the Crystal Windows or Web Report Viewer. To use this method you would first need to have created your Web service and noted the location and URL of the ASMX file, which we have already done.

Next, we need to create a project from within Visual Studio by selecting File → New → Project and from Visual Basic Projects, select Windows Application and specify a name (in the sample code, we have called this project consumer) and location for your project files.

Whenever you create a new project, a default form is added and to that form we need to add the Crystal Report Viewer. You can drag or draw the viewer onto your form and set any additional properties, methods, and events required. To bind the Crystal Report Viewer to your Report Web Service, you will

Chapter 6

need to set the ReportSource property in the load event (as shown subsequently using the URL from our earlier example):

```
CrystalReportViewer1.ReportSource =  
    "http://localhost/Chapter06/inventory_reportService.asmx"
```

With the ReportSource property set, you can treat this just like any other report, setting properties for the viewer and previewing the report, which is shown in Figure 6-11.

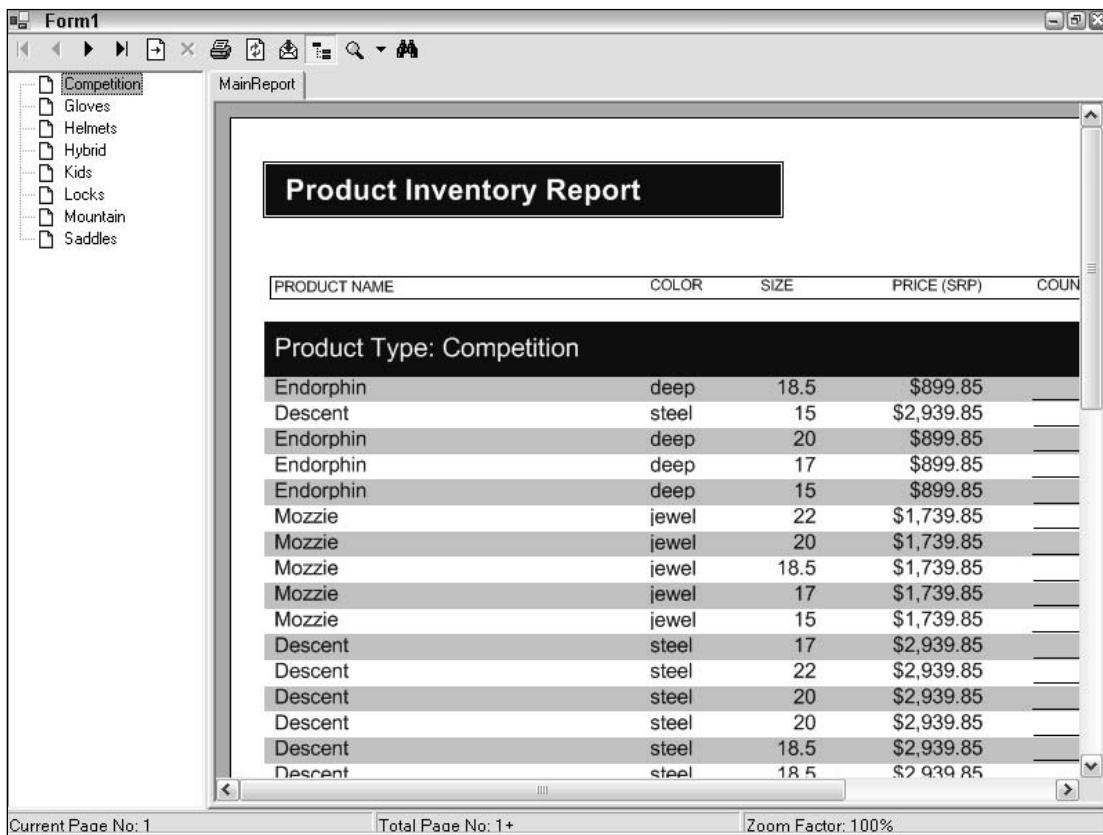


Figure 6-11

Internal Report Web Service

An internal Report Web Service refers to a Web service that has been added to your project as a reference. This method is sometimes also called the Proxy Method because every application that consumes a Web service has to have a way to communicate with the service when the application is running. Adding a reference to your Report Web Service creates a Proxy Class that in turn can communicate with the service and create a local copy.

Creating XML Report Web Services

To add the XML Report Web Service to your Web or Windows application, select Project → Add Web Reference to open the dialog shown in Figure 6-12.

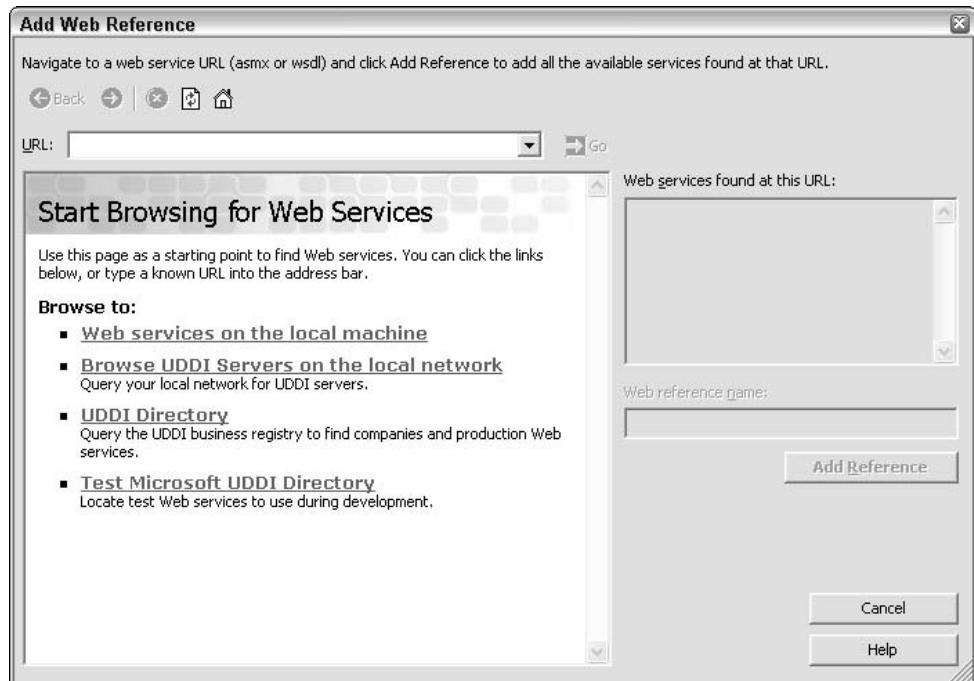


Figure 6-12

Using this dialog, enter the complete URL of your Report Web Service and click OK, and this will add this reference to your project in the Project Explorer under Web References.

To bind an internal Report Web Service to the Windows Crystal Report Viewer, you will again need to set the report source, only this time using the reference you have added instead of a URL. For example:

```
CrystalReportViewer1.ReportSource = New localhost.inventory_reportService()
```

From that point, all of the techniques you learned in Chapter 3, “Designing Reports,” and Chapter 4, “Report Integration for Web-Based Applications,” can be applied to the Report Viewer to customize how your report is presented.

Generic Report Web Service

To consume a report served through the generic Report Web Service, you will need to add the report from the Server Explorer to your project. In the Server Explorer dialog, under the node marked Crystal Services, navigate to the Server Files branch, where you should be able to see all of the available reports that are within the Samples path we looked at earlier.

To add a report to your project, simply drag and drop the report onto a Windows or Web form. From that point, you can add the appropriate Crystal Report Viewer to your form and bind the report to the viewer, as shown here:

```
CrystalReportViewer1.ReportSource = ServerFileReport1
```

All of the properties, methods, and events available in the Crystal Report Viewer can be used from this point.

Deployment Considerations

Report Web Services are deployed on a Web server and can be consumed by developers and users within your organization or externally, based on where you deploy the service itself and what access users have to the location you have selected.

When deploying Report Web Services (and Web applications in general), we have two deployment options. The first is the easiest and involves copying your project to the Web server for deployment. To use this method, you will need to open the project where your Report Web Service resides and select Project → Copy Project, which will open the dialog shown in Figure 6-13.

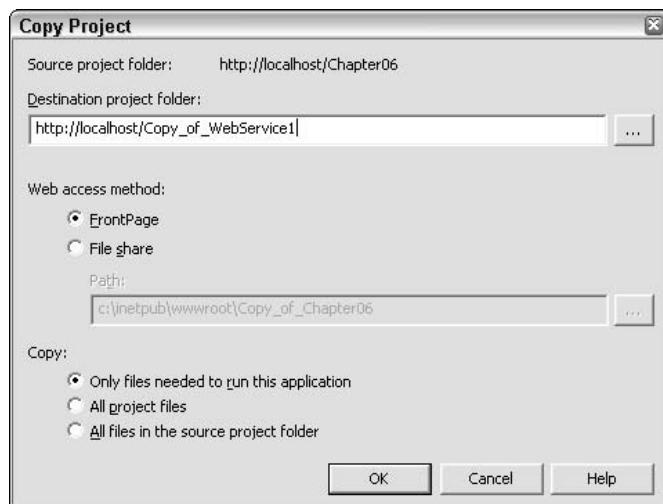


Figure 6-13

You will need to specify a folder location for your project, as well as a Web access method. If you select the FrontPage method, you must have the FrontPage Extensions installed and configured on the server where you want to deploy your Report Web Service. Choose the File share method if you have direct access to the server and just want to copy the files over.

At the bottom of the dialog, you will also need to select what files you want copied across to the Web server. You have three choices:

- Copy files needed to run this application**—will copy across all of the built output files and any files where the BuildAction property is set to Content
- All project files**—will copy across everything, including the output and source files, and so on
- All files in the source project folder**—will copy across all files within the folder, regardless of whether they are included in the project

The second method of deploying a Report Web Service involves creating a Web setup project that can be used to deploy your service on a local or remote Web server. To create a new Web setup project, select New → Project and from the folder marked Setup and Deployment Projects, select the icon for Web Setup Project and make sure you click the radio button for Add to Solution (instead of Close Solution).

We will call this project `WebServiceSetup`.

A new tab called File System on Target Machine will open. Right-click the Web Application Folder and select Add → Project Output. From the dialog that appears, shown in Figure 6-14, select Content Files, Primary Output, and Debug Symbols all at once using Ctrl-click.

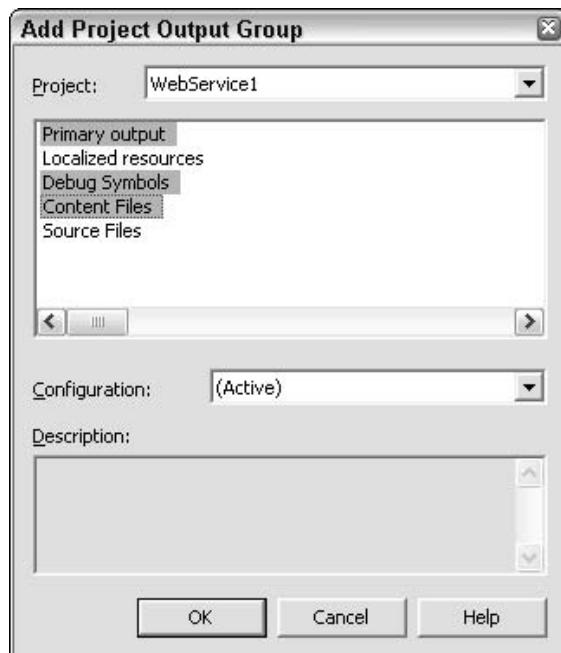


Figure 6-14

With these components added, save your project and select Build `WebServiceSetup` to create a Windows Installer Package (MSI) that will install your Report Web Service. If you are deploying this to a server that does not have Visual Studio .NET or the .NET Framework installed, you will need to install that before you install and deploy your Report Web Service.

Summary

In this chapter, we looked at XML Report Web Services and how they can be used to improve the exchange of information between organizations, cut down on development time, and extend simple reporting applications. We found that from the Visual Studio IDE, it takes only a few clicks to add a report and publish as a Web service.

With an XML Report Web Service created, the second half of the chapter focused on actually consuming and deploying XML Report Web Services. With that under our belt, it is time to take a look at working with .NET data.

7

Working with .NET Data

Up until now we have been creating reports directly off of tables in a relational database, but Crystal Reports .NET can actually be used to create reports from a wide variety of datasources, including ADO .NET, and it provides a number of tools for working with databases. In particular, there are a number of tools and techniques available within Crystal Reports .NET for working with SQL, allowing you to leverage your knowledge of SQL and any existing queries or reports that you may be using.

In this chapter, we will be looking at the way Crystal Reports .NET works with different data-sources and how it interacts with ADO .NET. This will include:

- Understanding data access
- Working with datasources
- Creating SQL commands and expressions within Crystal Reports
- Working with ADO .NET

At the end of this chapter, you will have an understanding of how Crystal Reports .NET interacts with different datasources, the options for working with these datasources, and using ADO .NET as a datasource for your report development.

The Sample Files

In the download files for Chapter 6 (`C:\Crystal .NET2003\Chapter06\`), you will find all of the datasets and reports used in this chapter:

- `Employee_Profile_Basic`—This version is used as the starting point in two examples in this chapter.
- `Employee_Profile_Table`—The same as the basic version with a second table added to the report.

Chapter 7

- Employee_Profile_SQLExp—The same as the basic version with an SQL Expression added to the report.
- SQLCommand—A report based on a Virtual Table created from an SQL command, discussed in “Defining Virtual Tables,” later in this chapter.
- Reporting_App_Dataset—An application that only contains a dataset.
- Reporting_App_ViewData—An application that displays an ADO .NET dataset.
- Reporting_App—A report that takes its data from an ADO .NET dataset.

If you have problems running these examples, please read the sections in this chapter relating to them.

Data Access with Crystal Reports .NET

Traditionally, Crystal Reports accessed data through two different methods—native connections and Open Database Connectivity (ODBC) connections. A native connection to a datasource was accomplished through a set of specialized .dll files and executables that were specific to your datasource. Over the years, Crystal Reports has teamed up with a number of database and application vendors to create native drivers for PC or file-type databases, relational databases, and various ERP (Enterprise Resource Planning) systems. The second data access method is through the ODBC layer, providing a common interface for interrogating relational databases. Regardless of where the data resides, ODBC provides a reliable, stable platform that can be used to develop drivers and data access methods.

With the integration of Crystal Reports into Visual Studio .NET, the native and ODBC drivers that were included with previous versions of Crystal Reports are no longer provided for use, and datasources are now accessed through one of the following methods:

Datasource	Description
Project data	Crystal Reports can leverage the ADO .NET Framework and report directly from the datasets that appear in your application.
OLEDB (ADO)	For datasources that can be accessed through OLEDB, including SQL Server, Oracle, and Microsoft Jet 3.51/4.00-accessible datasources (Access, Excel, Paradox, dBase).
ODBC (RDO)	For datasources that can be accessed through an ODBC-compliant driver (which is just about every other datasource). In addition to reporting from tables, views, and stored procedures, Crystal Reports .NET will also allow you to enter an SQL command to serve as the basis for your report (see “Working with SQL Commands and Expressions” later in this chapter).
Database files	Includes a number of file-type database formats, including Access, Excel, XML, and Crystal Field Definition files (TTX), as used with previous versions of Crystal Reports and bound reporting.
More datasources	Any other datasource supported by Crystal Reports .NET.

To help both new and existing report developers, the following section walks through the different types of data you may want to integrate into your reporting application.

Database Files

Previous versions of Crystal Reports could use a direct, native connection to create reports directly off of file-type databases, including dBase/Xbase, Paradox, and FoxPro, among others.

Through this direct connection, Crystal Reports extracted data without having to submit an SQL statement against a database server. With the ease of use, there was also a price. Because the data is held in a file and not on a database server, there is no opportunity to push the processing back down to the database server. Also, when working with these types of databases, the only join available between two or more tables was a left-outer join, meaning all of the information from the left-hand table will be read first, and any matching items from the right-hand table will also be shown.

As mentioned earlier, these native drivers (and the limitations that come with them) are not included with Crystal Reports .NET, apart from the direct drivers for Excel and Access. In order to create reports from these datasources, we have a number of options:

- Use an ODBC connection**—using a compatible ODBC driver to access your datasource
- Use an ADO .NET dataset**—create a dataset from your datasource

Create a custom data provider. For developers who have a specific data file format, you can create a custom data provider for your datasource. For more information on creating your own Custom Data Provider, check out the MSDN article at <http://msdn.microsoft.com/msdnmag/issues/01/12/DataProv/toc.asp>.

One type of native connection that is still supported is the direct connection to Microsoft Access databases and Excel spreadsheets. Both of these file types can be used as the datasource for your report without having to use ODBC.

Relational Databases

By far, the most popular data access method is through a native or ODBC connection to a relational database. The retail version of Crystal Reports that you would buy in a store ships native drivers for the most popular RDBMS, including DB/2, Informix, Oracle, and Sybase, among others. Most of these native drivers require that the standard database client be installed and configured before they can be used.

Again, these drivers are not available with Crystal Reports .NET, so you will need to look at connecting to these datasources through the following methods:

- Use an ODBC connection**—Uses a compatible ODBC driver to access your datasource.
- Use an OLEDB connection**—Uses a compatible OLEDB provider to access your datasource. Providers are available for SQL Server, Oracle, ODBC Drivers, and Jet 4.0, among others.
- Use an ADO .NET dataset**—Creates a dataset from your datasource.
- Utilize a custom data provider**—Currently, there is a custom data provider available for SQL Server and an Oracle provider in beta (available from the MSDN site) that allow you direct access to the database.

OLAP Data

OLAP data (sometimes called multidimensional data) can be accessed and used in your application through OLEDB for OLAP, a standard interface for accessing OLAP data, but unfortunately Crystal Reports .NET does not support OLAP reporting in this version. If you do have an existing report that shows an OLAP grid, this area will be blanked out when you first import your report.

Crystal Dictionaries, Queries, and Info Views

With previous retail versions of Crystal Reports, there were two separate tools designed to make report development easier. The first, Crystal Query, could be used to create Crystal-specific QRY files that contained SQL queries. You could then use these query files as the datasource for your report.

The second tool, Crystal Dictionaries, was used to create dictionaries (DC5, DCT) that served as a meta-data layer between your report and the database itself. Using a Crystal Dictionary, you could take care of all of the linking and joins for the user, reorganize and alias fields and tables, and add help text and data for browsing, among other things.

Unfortunately, none of these file formats is supported as a datasource for reports within Crystal Reports .NET. If you do have an existing report that uses any of these datasources, you will receive an error message and will be unable to use the report. If you wanted to create a report with similar features, you would need to base your report on the underlying database.

If you do need to work with complex SQL queries, Crystal Reports .NET provides the ability to use SQL Commands as the basis for your report, effectively cutting out the need to use Crystal Query files.

Currently, there isn't any way to work around Crystal Query files for creating a meta-data layer between the end user and the database itself other than using various third-party meta-data providers.

Other Datasources

In the past, Crystal Reports has included a number of drivers for nontraditional datasources, including SalesLogix Act!, Microsoft Exchange, Microsoft Logs, and more. Most of these datasources have had their own unique setup and configuration requirements, as they do not fit into the standard datasource categories that can be accessed through a native or ODBC driver.

Because the drivers for these datasources are not included with Crystal Reports .NET, you will need to find an alternative method of accessing this data, using a data provider.

So, in summary, Crystal Reports .NET supports the following datasources:

- Any database with an ODBC driver
- Any database with an OLEDB Provider
- Microsoft Access databases
- Microsoft Excel workbooks
- ADO .NET datasets
- Legacy recordsets (Classic ADO, CDO, DAO, RDO—which covers just about everything else)

Now that you understand the different ways Crystal Reports .NET can access data, we need to take a look at actually working with these datasources from within your report.

Working with Datasources

When working with data within the Report Designer, most of the options and functionality relating to databases and tables can be found in the Database menu, found under the main menu by right-clicking in your report (see Figure 7-1).

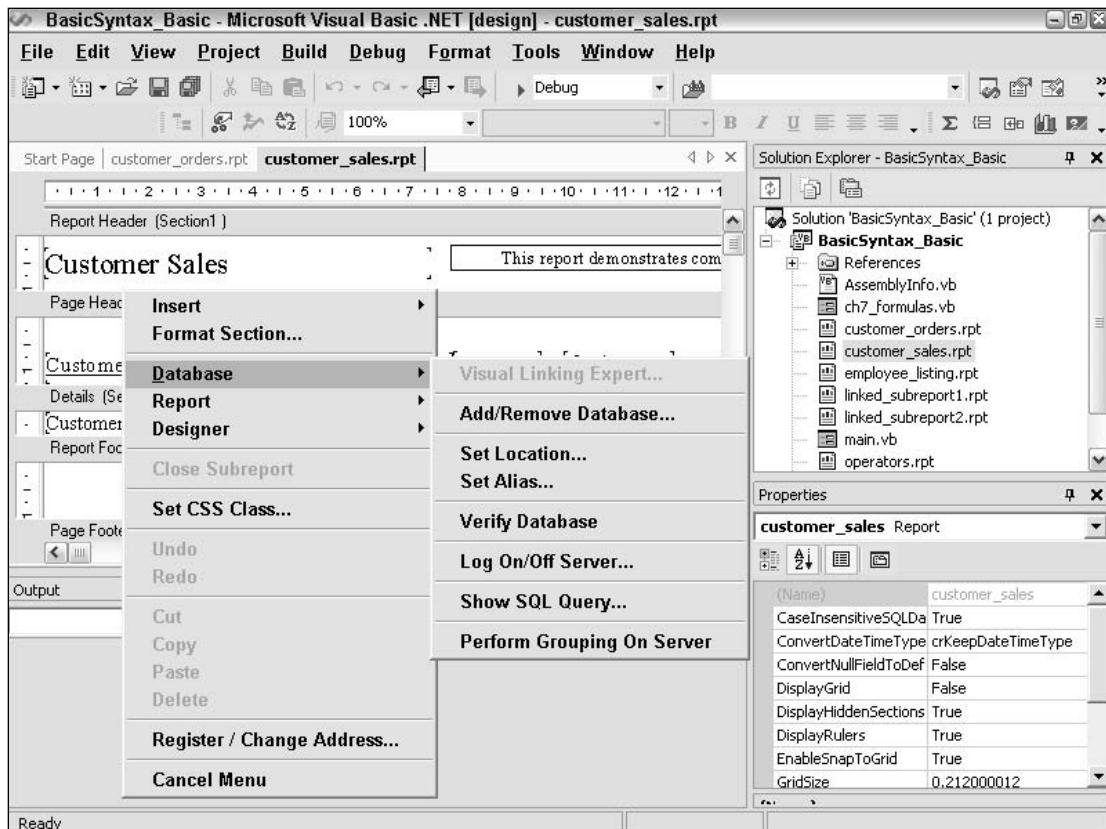


Figure 7-1

In the following sections, we are going to look at some of the most common tasks when working with data in our reports.

Setting Database Options

Crystal Reports .NET has a number of options that are specific to working with datasources and can be set once for the design environment. To see these settings, open any report, right-click, and select Designer → Default Settings → Database, which will open the dialog shown in Figure 7-2.

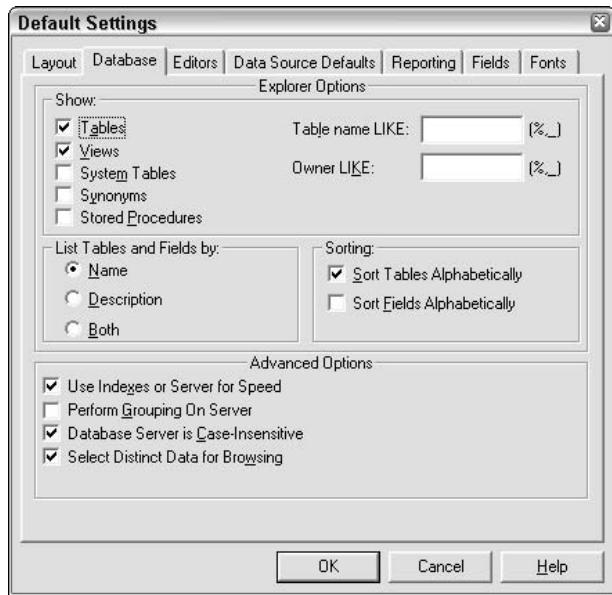


Figure 7-2

You can select the database objects you wish to show when creating a Crystal Report, including:

- Tables
- Views
- System Tables
- Synonyms
- Stored Procedures

This is where most developers get tripped up when working with stored procedures. They won't appear as an available datasource when you connect to your server *until* you turn the Show... Stored Procedure setting on.

You can also set a filter for database objects, using the Table name LIKE and Owner LIKE textboxes. Use the (%) symbol for multiple characters and the underscore (_) to indicate a single character. If you were looking for all objects owned by user DMCAMIS, the Owner LIKE text would be DMCAMIS%.

With the options in the middle of the dialog, select whether to list the tables and fields by their Name, Description, or Both, and use the checkboxes beside these options to sort table and field names alphabetically.

Finally, in the bottom of the dialog, under Advanced Options, select from the following:

- Use Indexes or Server for Speed**—Use existing database indexes or the database server itself for processing where a performance improvement could be gained.
- Perform Grouping on Server**—If you have created a summary report with none of the details showing and no drill-down capabilities, you can push the grouping of that report back to the server. This improves performance because Crystal Reports .NET doesn't have to get all the records and do the grouping itself. A GROUP BY clause will be inserted into the SQL that Crystal Reports generated.
- Database Server is Case-Insensitive**—By default, Crystal Reports .NET is case sensitive, meaning {Customer.Country}="ca" and {Customer.Country}="CA" would return different datasets. This setting eliminates that case sensitivity for SQL databases.
- Select Distinct Data for Browsing**—When browsing for data from pull-down or browse dialogs, this ensures that only a distinct recordset (no duplicates) is returned.

Because some of these changes relate directly to the database you are working with, you may need to log off and log back on for them to take effect.

Adding a Database or Table to a Report

When designing a report, you will need to add additional databases or tables from time to time as the need arises. In the sample reports included with this chapter, there is an Employee Profile (Employee_Profile_Basic) that lists employee names, birth dates, and hire dates. It does not list the employee's city; that information is held within another table that we are going to add to the report.

To start, open Employee_Profile_Basic in Visual Studio .NET by double-clicking the solution (.sln) file within that folder. Double-click employee_profile.rpt to open it in the Report Designer. Look at the four fields that are displayed in the report, and then look in the Field Explorer (View → Other Windows → Document Outline), where you will see the Employee table under Database Fields, as shown in Figure 7-3

We want to add the City to the report, but it isn't available from this table, so we are going to have to add the table that it is in. Right-click anywhere within the report, and select Database → Add/Remove Database, to open the Database Expert dialog (shown in Figure 7-4), which you can then use to add additional data structures to the report.

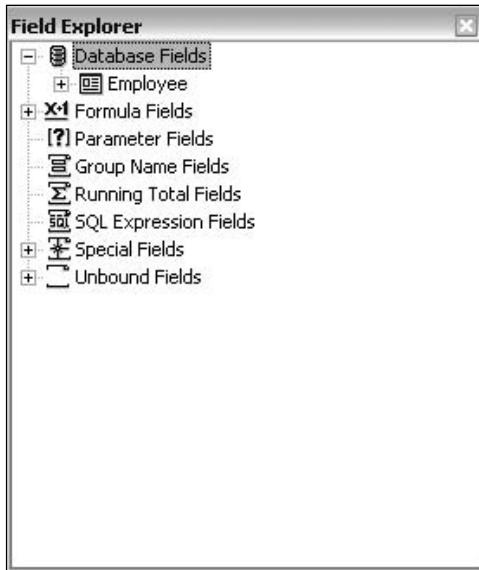


Figure 7-3

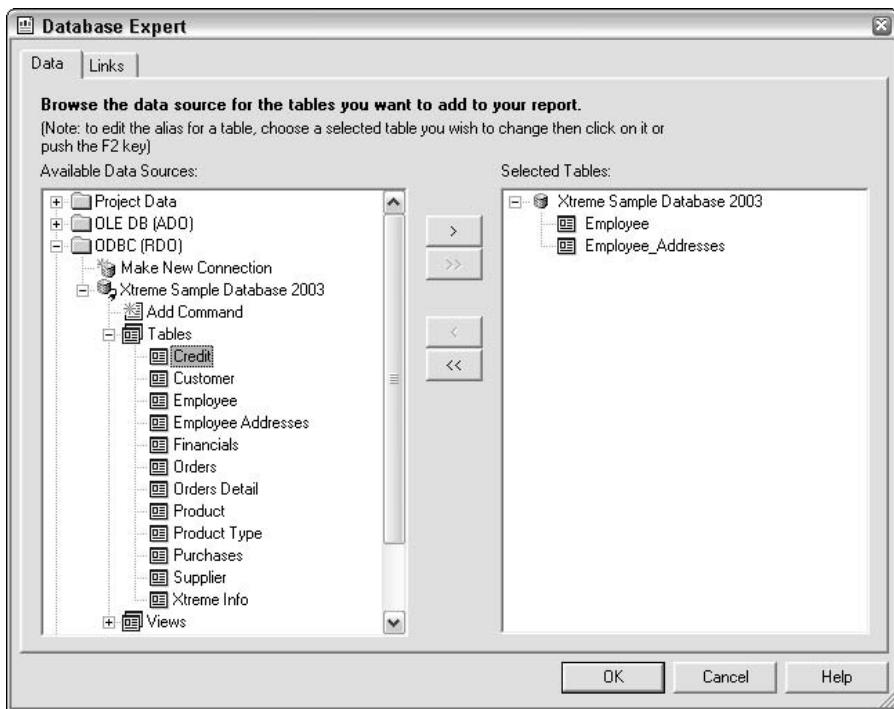


Figure 7-4

Locate the Xtreme Sample Database and expand the node to find the table you wish to add, in this case, Employee_Addresses. Select the table name and click the arrow icon to add it to the right window, which indicates that it has been added to your report. You can add further databases or tables here. When you have finished, select the Links tab, which allows you to specify the relationships between these tables, as shown in Figure 7-5.

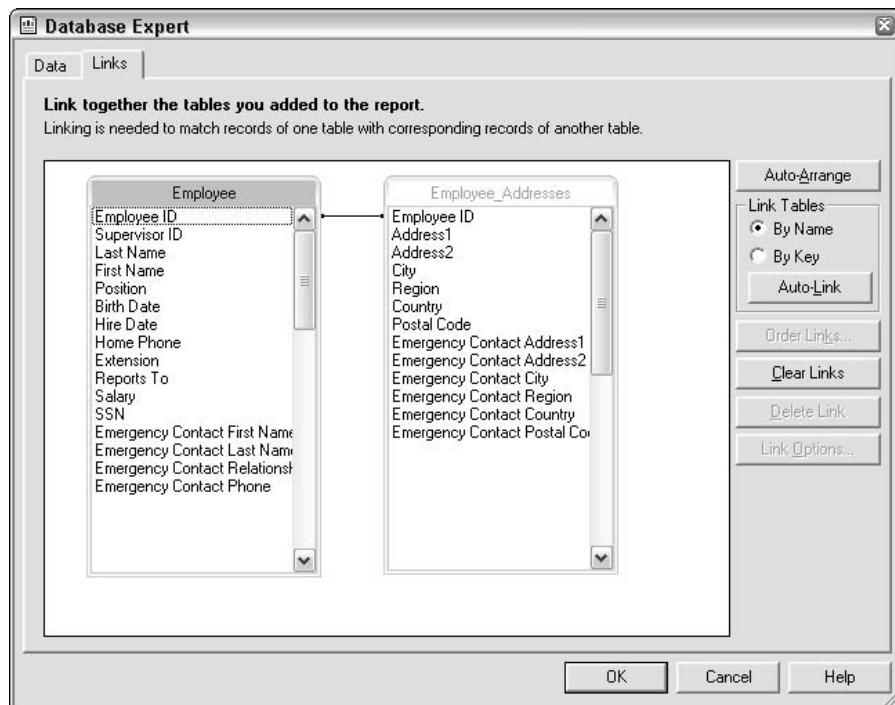


Figure 7-5

You may need to draw the link(s) to indicate the relationship between the new tables you have added to the tables currently in your report. On this occasion, the link between Employee and Employee_Addresses was automatically generated. By clicking the link, and then Link Options, we can set the options for the join types that these links represent (detailed in the next section). When you have finished, click OK to exit the Database Expert and return to your report design.

The first thing you should look for is your new table in the Field Explorer, shown in Figure 7-6.

If you expand the Employee_Address table, as shown in Figure 7-7, you can add some extra items to the report that you couldn't see before, for example, Country, Postal Code, and of course, City.

Chapter 7

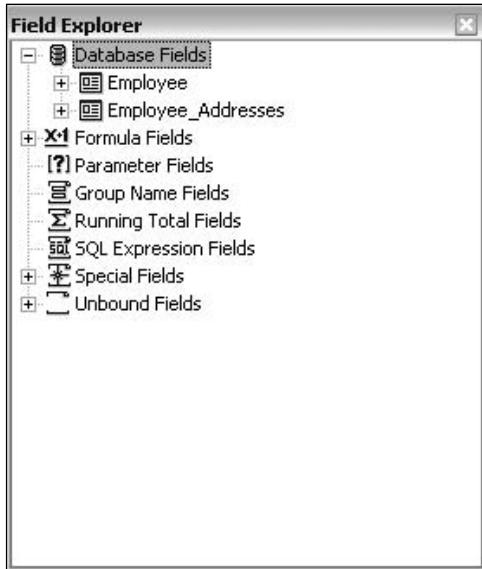


Figure 7-6

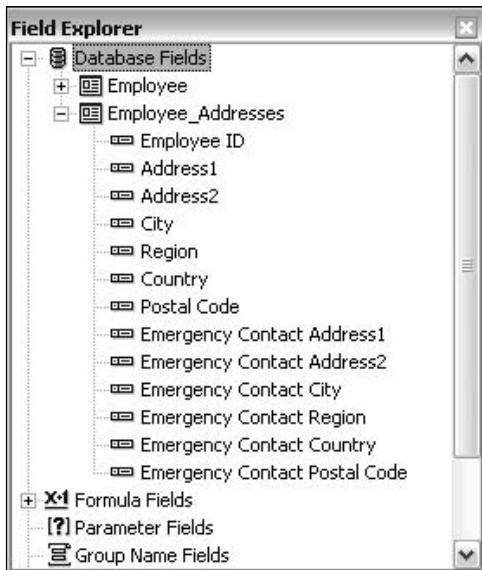
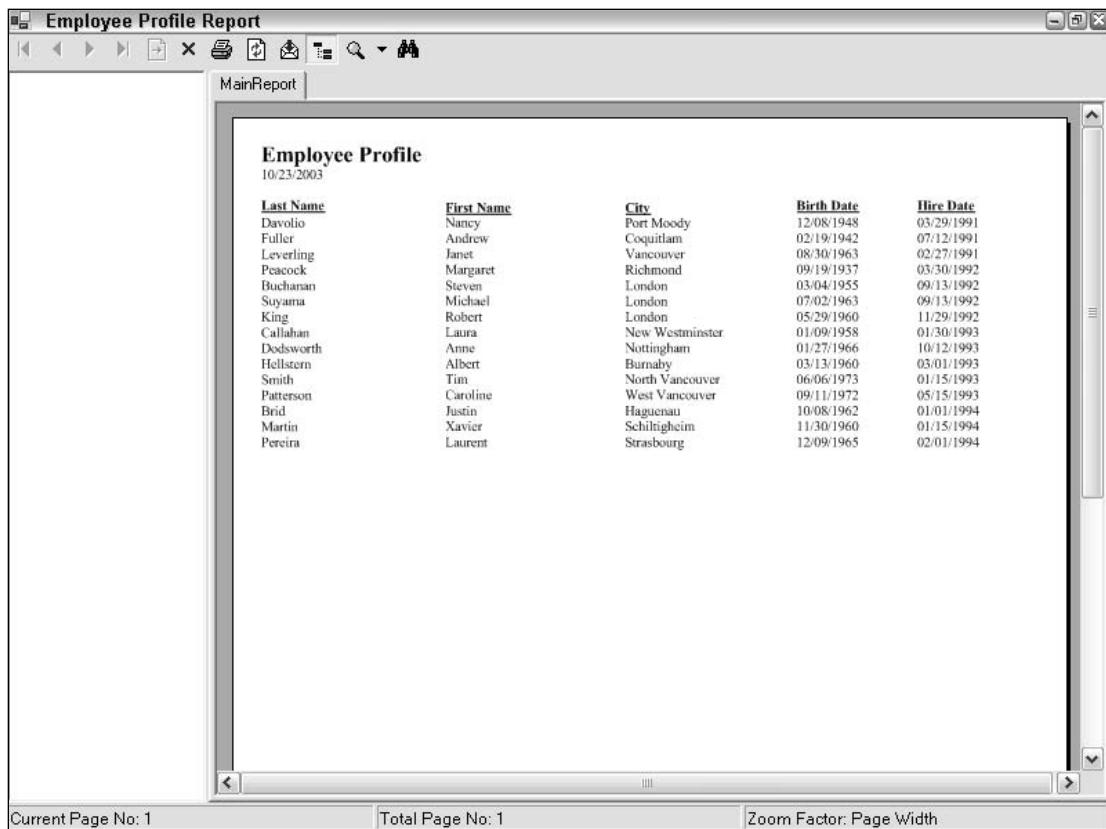


Figure 7-7

Drag City from the Field Explorer to the Details section of your report, and a label will automatically be added directly above it in the Page Header. You can now add additional fields as required; the report should now be similar to the Employee_Report_Table that is saved in the Chapter07 folder of the code download and shown in Figure 7-8.

**Figure 7-8**

You may want to adjust the linking in the Visual Linking Expert, or check the database schema for more information on how the tables should be joined together, which is described in the next section.

Using the Visual Linking Expert

Relational databases are usually split into a number of different tables; these tables can be joined back together to create complex queries. In Crystal Reports .NET, these joins are created by using the Visual Linking Expert to visually draw a line between two key fields and setting options on these links to indicate join types.

In addition to specifying database linking when you first add a datasource to your report, you can also invoke the Visual Linking Expert at any time by right-clicking your report and selecting Database → Visual Linking Expert from the right-click menu that appears, as shown in Figure 7-9 (this is the same dialog that you used a moment ago to set up the links).

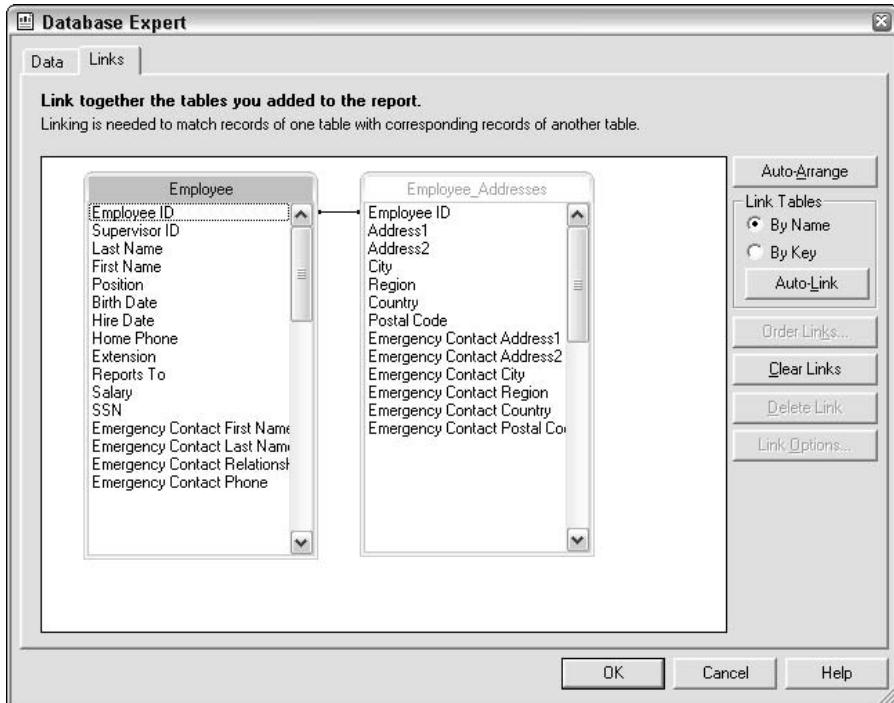


Figure 7-9

Using the dialog, you can draw links between the databases and tables in your report to indicate the relationship between each. To draw a line between two fields, imitate dragging the first field and dropping it on top of the second. You will know you have the field positioned correctly when your cursor turns into the shortcut icon.

If you make a mistake, you can remove a link by clicking the line to highlight it and pressing the delete key, or to clear all links, use the button of the same name on the right-hand side of the expert.

Our earlier example was very simple; in the Visual Linking Expert you can create multiple links between tables if your database schema requires them.

By default, Crystal Reports will join two SQL tables with an Equal join. To change the default join type, right-click directly on the line drawn between the two tables and select Link Options from the menu, shown in Figure 7-10.

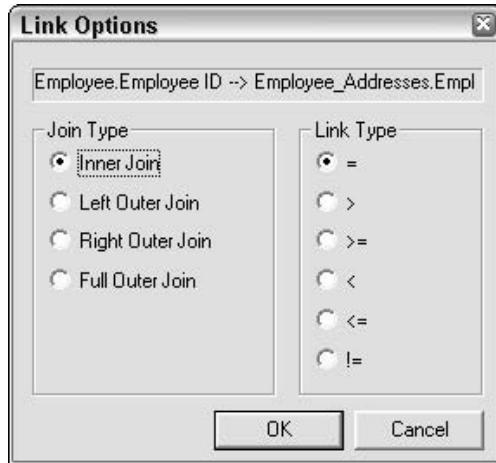


Figure 7-10

Using the Link Options dialog, select a join type for this link from the following list:

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

You can also select an operator to work with the join type you have selected, including:

Operator	Description
=	Equal To
>	Greater Than
≥	Greater Than or Equal To
≤	Less Than
≤=	Less Than or Equal To
!=	Not Equal To

At any point you can click the Auto-Arrange button to arrange the Visual Linking Expert layout for readability, but sometimes you will get better results if you position the tables yourself.

If it's still open from the last example, take a look at `employee_profile.rpt` in Crystal Reports .NET. The report has been created from the `Employee` and `Employee_Addresses` tables. To see the SQL that Crystal Reports .NET has generated, right-click the report, and select Database → Show SQL Query:

```
SELECT `Employee`.`Last Name`, `Employee`.`First Name`, `Employee`.`Birth Date`,  
`Employee`.`Hire Date`, `Employee_Addresses`.`City`  
FROM `Employee` `Employee` INNER JOIN `Employee Addresses` `Employee_Addresses`  
ON `Employee`.`Employee ID` = `Employee_Addresses`.`Employee ID`
```

The fields you select in your report control the contents of the SELECT statement, but it is the links that control the FROM clause. When working with multiple tables or a large database, your database administrator should be able to give you some guidance on how the tables should be arranged and joined together.

If you find working with links in Crystal Reports .NET difficult, you can always use an SQL command as the datasource for your report, and perform any joins in the SQL statement you write.

Verifying Database Structures Used in Your Report

As your database structures evolve and change, Crystal Reports you have created from these structures may no longer work due to differing field names and types. To ensure that the changes made in the database are reflected and accounted for in your existing reports, you will need to verify the database that they were created on by selecting Database → Verify Database from the main right-click menu.

If you have databases or tables in your report that are not used, you may receive the message "Verify files in report that are not used?" Click Yes to proceed.

At this point, Crystal Reports .NET will run through the data structures in your report and verify that nothing has changed. If all of the data structures are unchanged, "The database is up to date" will be displayed on your screen.

If anything has changed in the data structures, you will receive a message informing you of this, and that Crystal Reports .NET is proceeding to fix the report (shown in Figure 7-11).



Figure 7-11

If Crystal Reports .NET finds simple changes, like a database field that has been extended or a decimal place that has changed, it will simply update its version of the data structures and display the message "The database is up to date."

If Crystal Reports .NET finds a major change (like a field name missing, or a changed field type) it will open a Map Fields dialog. A list of unmapped fields will appear in the upper left-hand corner of this

dialog. These are fields that are currently in your report that Crystal Reports .NET could not find when it attempted to verify the underlying data structure. To resolve any mismatched fields, select a field out of the dialog that has Report Fields at the top, locate its counterpart in the list on the right-hand side, and click Map.

If the type of the field has changed as well as the name, uncheck the Match Type option to show all fields.

When you have finished mapping all of the fields that were not found in the verification of the data structures, you can return to your report design and Crystal Reports .NET will use these mapped values in place of the missing fields.

Changing a Database Location

Another handy feature is the ability to change the location of the database that your report uses. For example, you can design a report on your test database, and then later point it to a production version. To change the location of the database in your report, right-click and select Database → Set Location and from the dialog shown, select the database or table you want to point to.

Using the Data Explorer tree on the right, locate the datasource and database or table you wish to change the location to, highlight, and click the Replace button.

If you are using multiple databases or tables in your report and subreports, there is a checkbox in the bottom left-hand corner with the option of Apply this change to subreports. Turning this option on will change the database location for all other subreports as well.

If the data structures are different between the old database or table and the database in the new location you have selected, the Map Fields dialog will appear (Figure 7-12), and you must map any unfound fields in your report to fields in the new database structure.

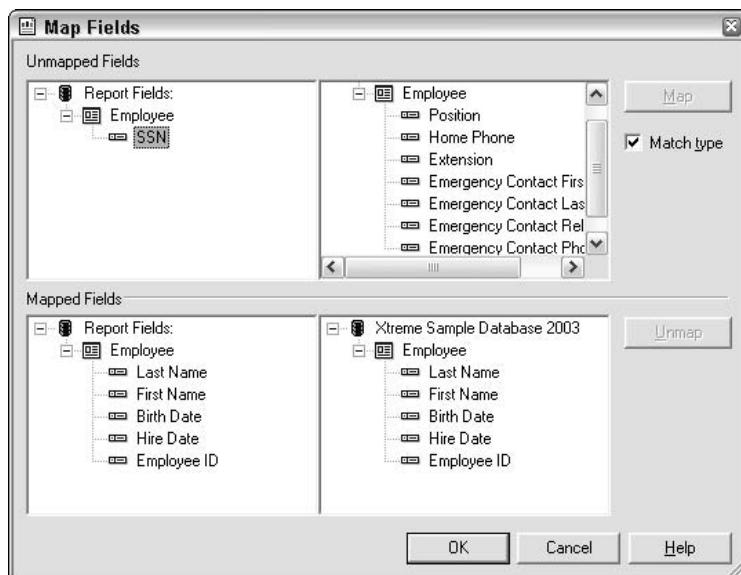


Figure 7-12

Setting a Database Alias

Aliases are used when you need to reference a table in a report more than once. A common example would be where you had an employee table with a supervisor ID that was also the employee ID of the supervisor. To get Crystal Reports .NET to reference the same table, you would need to add it to the report a second time, and give it an alias like EmployeeSupervisor.

To set a database alias, open the Database Expert by right-clicking and selecting Database → Add/Remove Database. In the window on the left, find and then double-click the database or table you want to apply the alias to, and the dialog shown in Figure 7-13 will open.

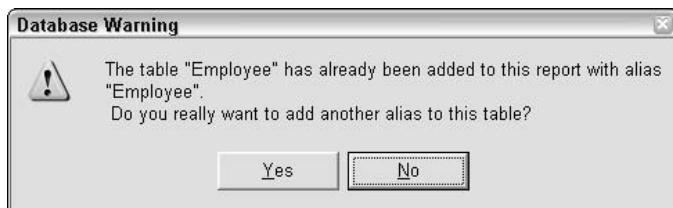


Figure 7-13

Select Yes, and enter the new alias for the database in the Alias Name dialog, shown in Figure 7-14.

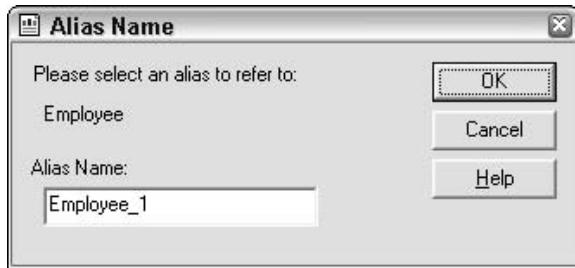


Figure 7-14

You will see your new alias in the right-hand window of the Database Expert, with the alias you specified in the previous dialog, shown in Figure 7-15. You can edit this name by selecting the table in the Selected Tables window, and pressing F2.

This will only change the alias of the datasource within Crystal Reports .NET and will not touch the underlying SQL statement. Selecting OK will accept this and present you with the Visual Link Designer, as seen in Figure 7-16, which you can use to change the links between the different tables, as we described earlier in this chapter.

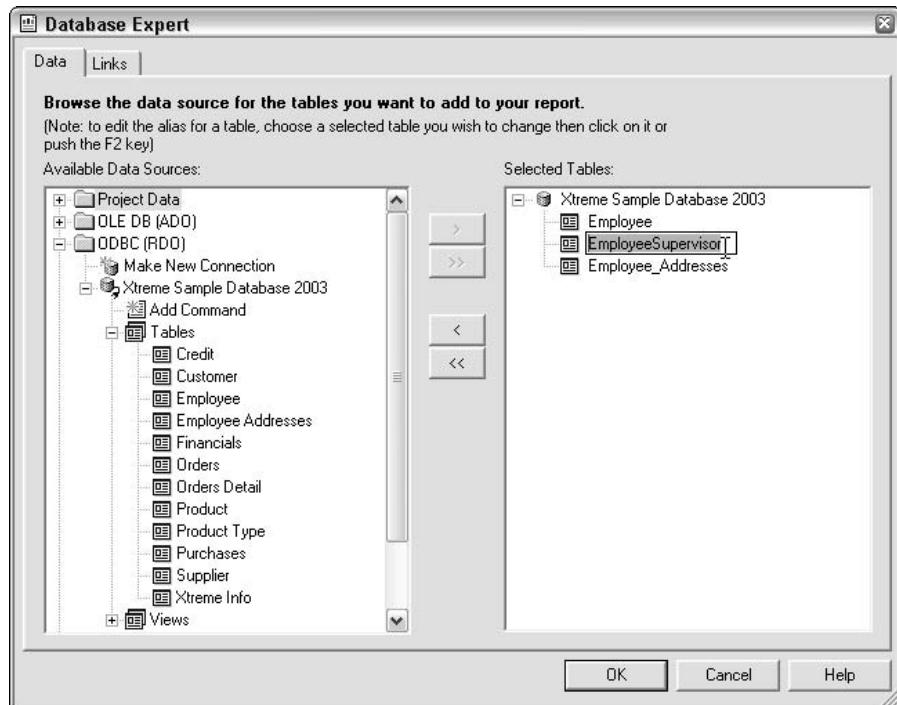


Figure 7-15

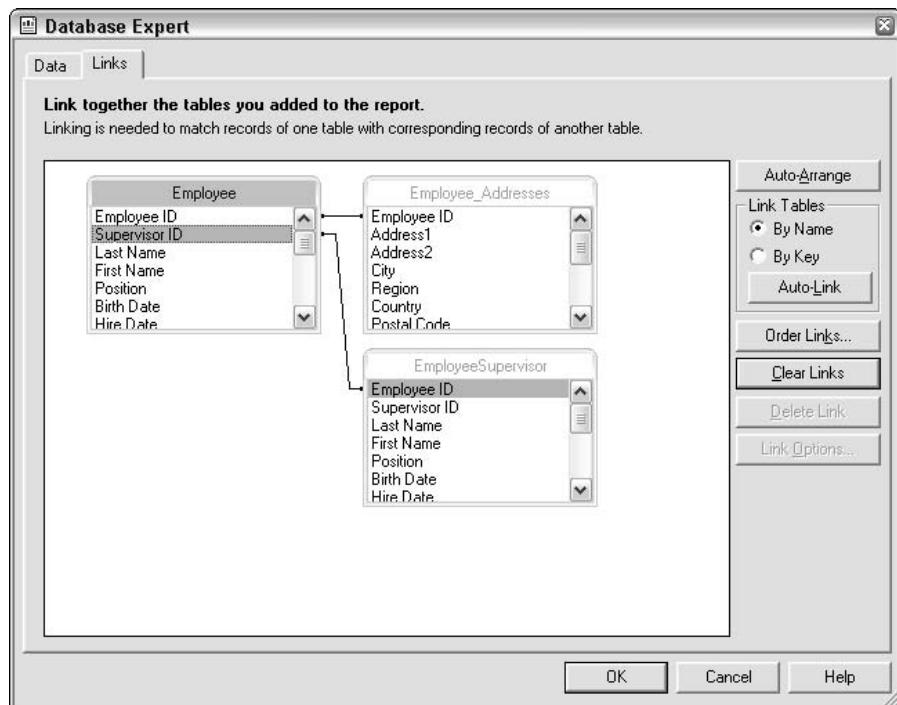


Figure 7-16

Working with SQL Commands and Expressions

In addition to reporting directly from database tables, views, and stored procedures, you also have the ability to use custom SQL commands as the datasource for your report. Using this method, you can create a *virtual table* that contains all of the fields you want to use. This functionality offers a flexible alternative to using Crystal Reports' own database and linking functionality, and can help you reuse the investment you have made in other report tools, or existing SQL statements. SQL Expressions are used within a report to create new values to display; for example, projected sales figures can be generated by applying a mathematical expression to the previous year's sales.

Defining Virtual Tables

To see this feature in action, create a new Visual Basic .NET Windows Application within Visual Studio .NET called `SQLCommand`. (This project is available in the code download at the location `C:\Crystal.NET2003\Chapter07\SQLCommand.`) Select Project → Add New Item and then choose Crystal Report from the available templates. Call the file `sql_command.rpt`, as shown in Figure 7-17, and click Open. We will first step through the setup of a basic report again.

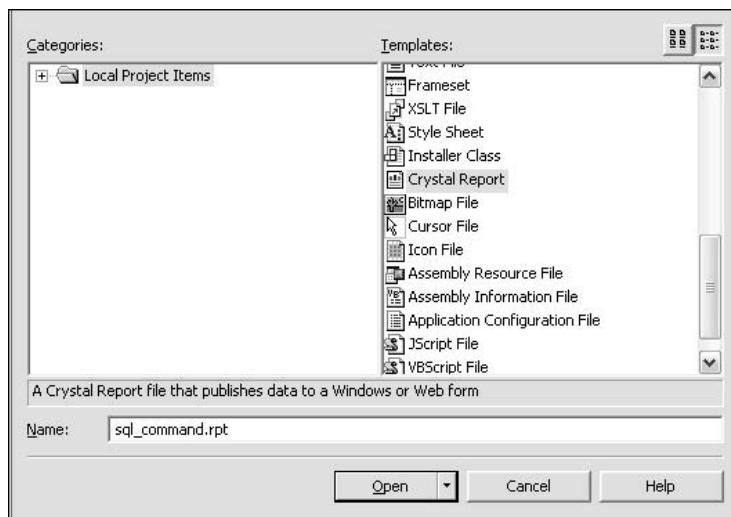


Figure 7-17

This will open the Crystal Report Gallery and allow you to select an expert to help you get started. In this example, we are going to select the Standard Report Expert, but SQL commands can be used with any of the experts listed.

The first step of the Standard Report Expert is selecting the datasource for your report; double-click the node for ODBC (RDO) and select Xtreme Sample Database 2003 as our sample datasource. This will open the dialog shown in Figure 7-18 where you can double-click Add Command to do just that.

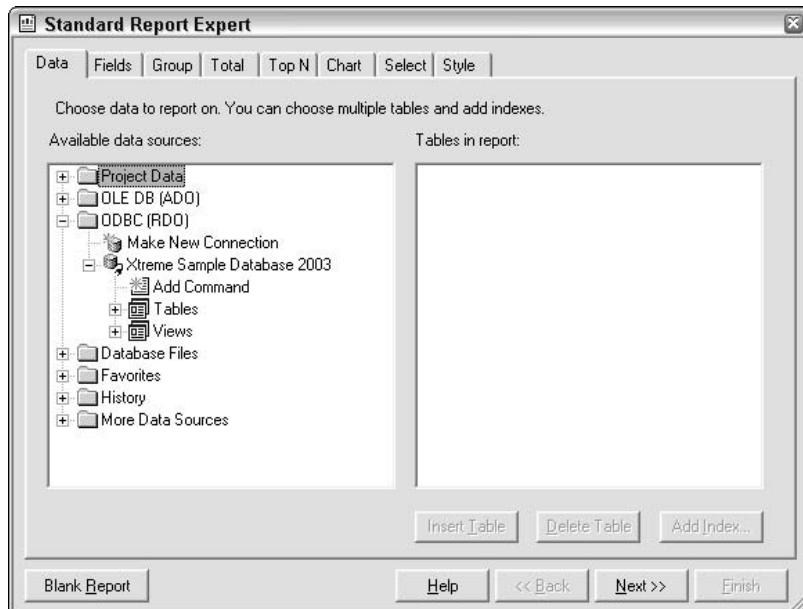


Figure 7-18

When adding an additional datasource to an existing report using Database → Add/Remove Database, the option to add an SQL command is also available.

When you select Add Command, another dialog will open and allow you to enter an SQL statement to serve as the datasource for your report. Enter `SELECT * from Customer` in the window, as shown in Figure 7-19.

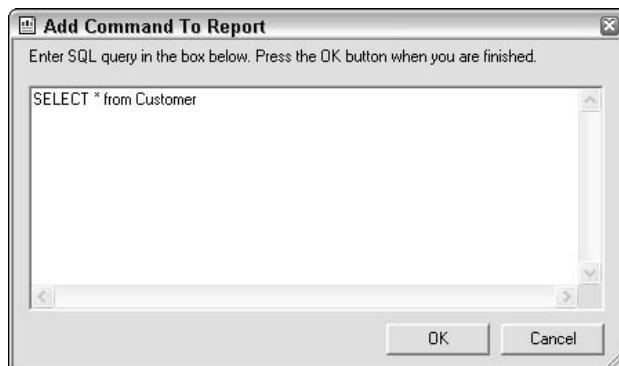


Figure 7-19

Chapter 7

Click OK, and Crystal Reports will treat the results of this query as a virtual table. You can now use the fields you have specified in your select statement in your report.

Click Next to move on to the Fields tab, and where you would normally see a table to select data from, you will see your command. It behaves just like a table in this dialog, so click the + to expand the command, as shown in Figure 7-20 and you will see the results of your SQL command, in this case the Customer table.

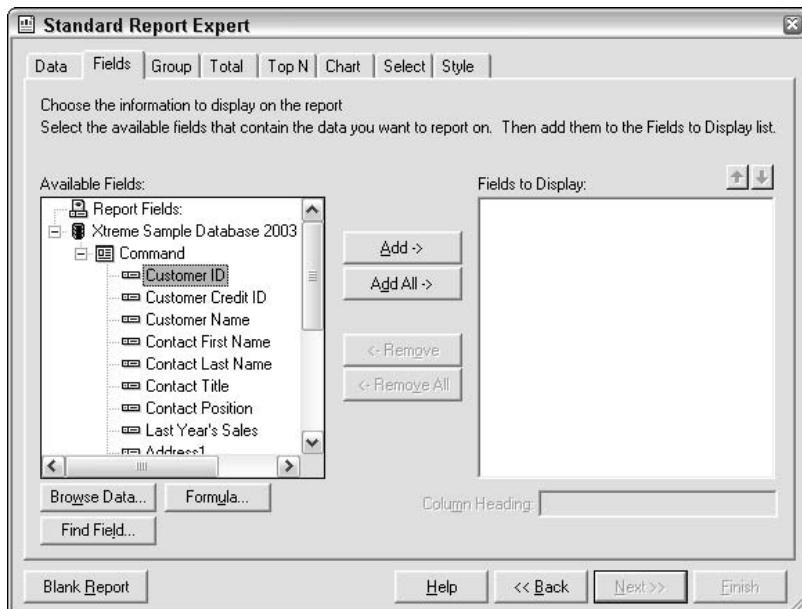


Figure 7-20

Now select the fields you want to see on your report, which in this case are:

- Customer Name
- Contact First Name
- Contact Last Name
- City
- Region

That's all you need to do to make a basic report based on an SQL command, so click Finish to generate the report, and your Report Designer should show these fields in the Details section of `sql_command.rpt`. You know how to preview your report now, but here is a reminder of the basic steps:

- Drag the `CrystalReportViewer` (under Windows Forms in the Toolbox) to `Form1.vb`
- Drag `ReportDocument` (under Components in the Toolbox) to `Form1.vb`

- Double-click the CrystalReportViewer that sits on your form to generate the procedure that loads your report when you run the report
- Insert the following code in the procedure:

```
CrystalReportViewer1.ReportSource = New sql_command()
```

When you run your report, it should look something like the report in the Figure 7-21, and you now know how to create a report based on a Virtual Table created from an SQL command.



Figure 7-21

Troubleshooting SQL Commands

After you have worked with SQL commands in Crystal Reports .NET, you will realize that the error messages returned aren't always the most informative. A lot of the problem with this has to do with the error messages that are returned from the database, as opposed to Crystal's own error handling and messaging.

In an effort to keep errors to a minimum, take the following points into consideration:

- ❑ The dialog to enter an SQL statement has no syntax checker or editing tools. To ensure your SQL statement will work, test it first in your own SQL query tool (SQL*PLUS, Query Analyzer, Microsoft Query).
- ❑ The SQL statement that you enter must include a `SELECT` statement and return a result set, and cannot contain any data-definition or manipulation commands.

Creating SQL Expressions

One way to improve report processing in Crystal Reports .NET is to use SQL expressions in your report instead of formulas written in Crystal Reports. These SQL expressions are passed back to the database and all of their processing occurs there. To understand how to use SQL expressions in a report, open `Employee_Profile_Basic` from the code download.

Open `employee_profile.rpt` in the form designer, locate the section of the Field Explorer marked SQL Expression Fields, and right-click directly on the section. From the right-click menu, select New... and enter a name for your SQL expression (Figure 7-22).



Figure 7-22

In this case we are going to create an SQL Expression called New Salary that will show the effect of a 6% increase on the employee's salary. Once you have given your SQL expression a name and clicked OK, the SQL Expression Editor will open, which you can use to create an SQL expression using the available fields, functions, and operators, as shown in Figure 7-23.

If the SQL Expression Editor looks familiar, this is because it is really the Crystal Reports Formula Editor in a different guise. The standard functions and operators have been replaced with SQL functions and operators.

The SQL Expression Editor is not that flashy on features, but does include a syntax checker. Enter your expression by double-clicking the field you want to work with (Salary), and then do the same for the multiply operator, and finally enter 1.06 to represent the 6% increase in salary. To check the syntax of your expression, click the X+2 button located in the toolbar, and you will hopefully see the dialog shown in Figure 7-24.

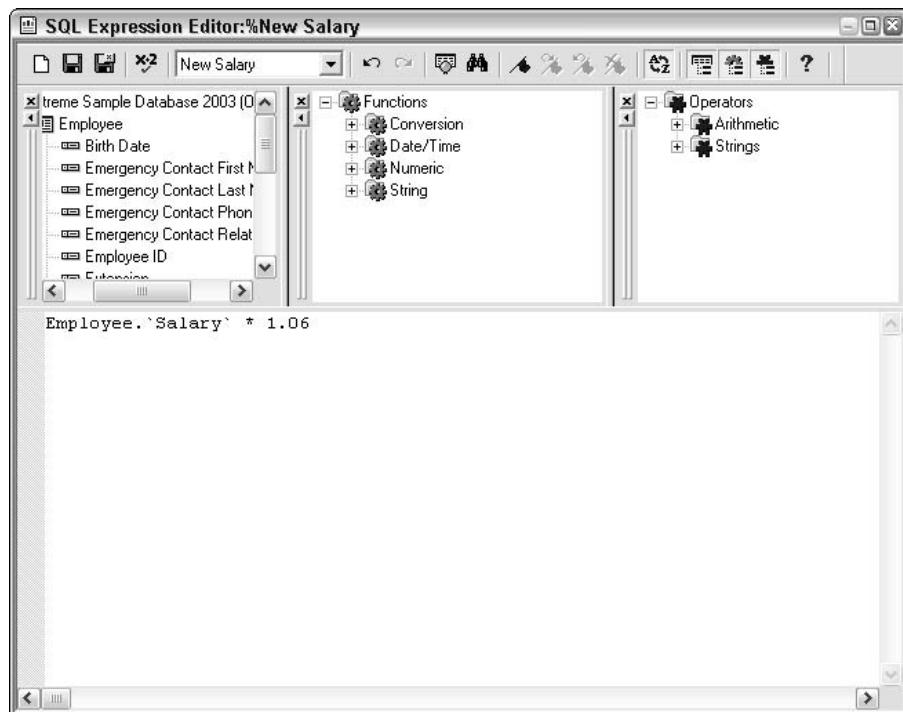


Figure 7-23



Figure 7-24

When you have finished editing the SQL expression, click the Save and Close icon in the upper left-hand corner to exit.

Sometimes when the syntax checker passes your expression, complex expressions may still fail when executed, because unfortunately the checker doesn't know everything; however, it will point out any basic errors or typos.

Only use the functions and operators supported by your database. If in doubt, consult with your database administrator on the correct usage of syntax.

Your SQL Expression should now appear in the list in the Field Explorer and you can drag and drop this field onto your report. The field should appear just like any other field shown on your report, showing that the SQL Expression itself has been added.

You can see that the SQL Expression can be identified as a % character and precedes the name, New Salary.

When your report runs or is previewed, this calculation will occur on the database server and the results are returned to your report, just like any other database field. This report you have created is the same as Employee_Profile_SQLEx.rpt, which is available in the code download for this book.

Working with ADO .NET

ADO .NET is a new data interface standard introduced with the .NET Framework that provides access to many different types of data, including SQL Server, OLEDB-compliant datasources, and XML. Using ADO .NET, all of the data access and manipulation is handled by a number of individual components that work together to select from, insert into, and update datasources.

When working with ADO .NET, the resulting data can be stored in a special structure called a dataset, which can be used by the application itself and as the datasource for a Crystal Report.

A dataset is a disconnected, in-memory data collection. Applications use ADO .NET to populate the DataSet object, which you can then write to, search, copy from, and so on. To run reports that show this list of information, we don't actually need to go back to the database again, as the information is held within the dataset.

We can design the report based on the structure of this dataset, and when the report is run, we point the report to this data, which is used to preview and print the report. This provides a definite performance advantage and represents the best practice for writing reports that use application data. In the following sections, we are going to briefly cover how to create an ADO .NET dataset and then use it as the data-source for a report.

An ADO .NET Dataset

To create an ADO .NET dataset, we need to use the ADO .NET Dataset Designer. To invoke the Designer, create a new Windows application called Reporting_App. Select Project → Add New Item and select DataSet from the list of available templates, as shown in Figure 7-25.

Set the name of the dataset to CustomerOrders .xsd, and then click Open to see the Dataset Designer with a blank view. You should notice that in the Server Explorer the CustomerOrders dataset that you have just created has appeared.

To start building your dataset, you will need to specify where the underlying data comes from. On the left-hand side of the Visual Studio .NET IDE, locate the Server Explorer and find the section marked Data Connections. This toolbar contains all of the datasources that can be accessed, including the Northwind database that we will use in this example (accessing this database is discussed earlier in Chapter 3, “Designing Reports”).

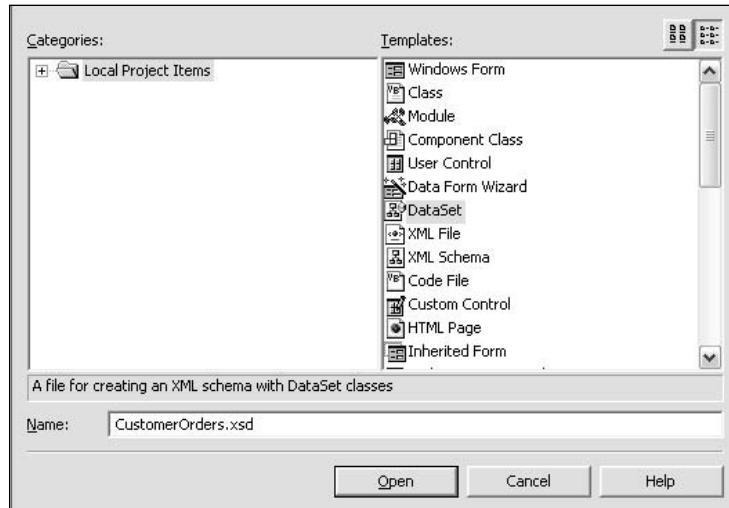


Figure 7-25

If your datasource is not listed, right-click Data Connections and select Add Connection from the right-click menu. To set up a new Data Connection, you will need to specify a provider and the appropriate server name and credentials for your datasource.

The Northwind database that we want to use should be available. The tables and fields within the datasource should appear below its node.

To build a dataset, you can simply drag and drop the required tables from your database connection onto the dataset design surface and specify the relationships and keys between the datasources. In our example, we are going to use the `Customers` and `Orders` tables from the Northwind database, so add these tables.

The relationship between the two tables is on the `CustomerID`, so to create this link you can click and drag from the `CustomerID` in the `Orders` table to the `CustomerID` in the `Customers` table. This action opens the Edit Relation dialog (also opened by right-clicking an existing link and selecting Edit Relation), in which you select the parent and child elements from the two drop-down lists. Our parent element should be set to `Customers`, our child element should be set to `Orders`, set the key to be `CustomerOrdersKey1`, and both of the key fields should be set to `CustomerID`.

Select OK to see the customers and orders relationship. Make sure that `CustomerID` and `OrderID` are the only key fields in each of the tables, because we want the tables in the dataset to match the tables in the Northwind database.

You can view the XML that describes this relationship and data by clicking the XML tab at the bottom left of the dataset design window. You can check that your relationship has been entered correctly, by comparing the three fields that can be found at the bottom of your XML schema with these three fields here; they should match:

```
</xs:element>
</xs:choice>
</xs:complexType>
<xs:unique name="CustomerOrdersKey1" msdata:PrimaryKey="true">
    <xs:selector xpath=".//mstns:Customers" />
    <xs:field xpath="mstns:CustomerID" />
</xs:unique>
<xs:unique name="CustomerOrdersKey2" msdata:PrimaryKey="true">
    <xs:selector xpath=".//mstns:Orders" />
    <xs:field xpath="mstns:OrderID" />
</xs:unique>
<xs:keyref name="CustomersOrders" refer="CustomerOrdersKey1">
    <xs:selector xpath=".//mstns:Orders" />
    <xs:field xpath="mstns:CustomerID" />
</xs:keyref>
</xs:element>
</xs:schema>
```

You can manually edit the XML, although it is much easier to stay within the bounds of the development environment, and edit the dataset using the Properties window. Information on creating complex datasets is available in *Fast Track ADO .NET* (1-86100-760-4) from Wrox Press.

Now that you have finished creating your dataset, select Build Solution from the Build menu to build your dataset and generate the database object for this dataset. Create a copy of this project so we can use this dataset in a report that we will build later in this chapter. We will now go on to display this dataset in a **DataGridView** on a Windows form.

Viewing the Contents of a Dataset

Open `Form1.vb` in design mode, and from the Windows Forms section of the Toolbox, drag a **DataGridView** to the form. Just for presentation, add a label to the top of the form saying **The CustomerOrders Dataset**.

Next add an **OleDbDataAdapter** to the form by dragging and dropping one from the Data section of the Toolbox. This action opens the Data Adapter Configuration Wizard, shown in Figure 7-26, which will help you step through the process of setting up your data adapter. The first thing you will have to do is select your data connection. Make sure this is the same as the datasource you built your dataset from, in this case, the Northwind database.

If it is not the default, enter the details for your Northwind database. Click New Connection to open the Data Link Properties window. Enter the location of your SQL Server and the security settings, and select Northwind from the list of available databases. Test the connection, and if all is good, click OK to continue setting up the **OleDbDataAdapter**.



Figure 7-26

The next dialog allows you to choose a query type to decide how the data adapter will query the database. We will use SQL statements for this purpose, so select Use SQL statements. Click Next to get to the Generate the SQL Statements dialog, and then click Query Builder to generate your statement. This opens the Query Builder, and in front of it, the Add Table dialog, as shown in Figure 7-27.

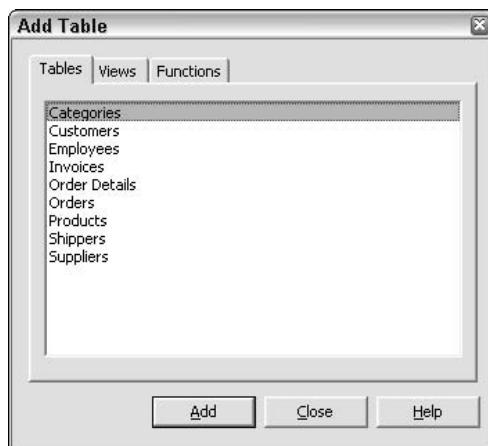


Figure 7-27

Chapter 7

Add the Customers and Orders tables that make up our dataset and click Close, and the SQL query will start to be built. Then simply check the (All Columns) boxes in both of the tables to complete our query (Figure 7-28).

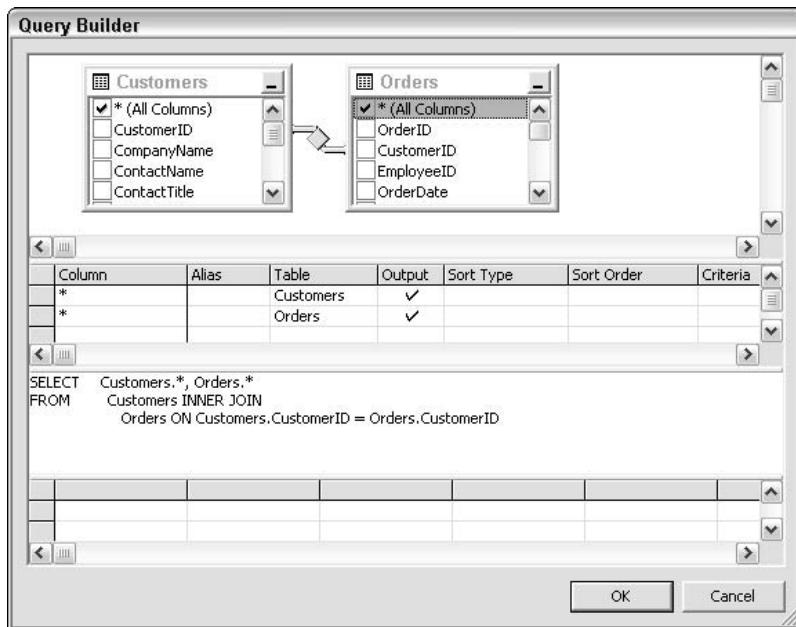


Figure 7-28

Click OK and then Finish, to complete configuration of the data adapter.

Now that the datasource has been defined, add a data grid (if you haven't already), double-click the form (not in DataGrid1) to open up the code designer for Form1.vb. This action creates a procedure called Form1_load, which fires when the form is loaded. Insert the following code into this procedure:

```
Private Sub Form1_Load(ByVal sender As System.Object,  
                      ByVal e As System.EventArgs) Handles MyBase.Load  
    Dim CustomerOrders As New DataSet()  
    OleDbDataAdapter1.Fill(CustomerOrders)  
    DataGrid1.SetDataBinding(CustomerOrders, "Customers")  
End Sub
```

This code will use the data adapter we have just set up to fill the dataset with information from the Customers table in the Northwind database, and display it in DataGrid1.

Working with ADO .NET requires two namespaces: `System.Data`, and because we are using OLEDB, the `System.Data.OleDb` namespace as well. Add the Imports statements to the top of Form1.vb.

```

Imports System.Data
Imports System.Data.OleDb

Public Class Form1
    Inherits System.Windows.Forms.Form

```

Run the code and your application should open and display the dataset it has been loaded with, providing you with access to data from the *CustomerOrders* dataset, as shown in Figure 7-29.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Repres	Obere Str. 57	Berlin
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Repres	Obere Str. 57	Berlin
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Repres	Obere Str. 57	Berlin
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Repres	Obere Str. 57	Berlin
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Repres	Obere Str. 57	Berlin
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Repres	Obere Str. 57	Berlin
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la C	México D
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la C	México D
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la C	México D
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la C	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2	México D
AROUT	Around the Horn	Thomas Hardy	Sales Repres	120 Hanover	London
AROUT	Around the Horn	Thomas Hardy	Sales Repres	120 Hanover	London
AROUT	Around the Horn	Thomas Hardy	Sales Repres	120 Hanover	London
AROUT	Around the Horn	Thomas Hardy	Sales Repres	120 Hanover	London
AROUT	Around the Horn	Thomas Hardy	Sales Repres	120 Hanover	London
AROUT	Around the Horn	Thomas Hardy	Sales Repres	120 Hanover	London

Figure 7-29

The application you have created should be the same as *Reporting_App_ViewData*, which is available in the code download. Now we will go on to using the dataset as the datasource for a report.

Creating a Report from an ADO .NET Dataset

The *CustomerOrders* dataset for the database object we have built will be populated when the application is started. When designing a report based on this dataset, we cannot browse the content of any of the fields as the dataset hasn't been populated at this point, but if you do have any questions about what type of content the fields could contain, view the database to find out, or use a viewing application like the one we have just built.

To create a report based on an ADO .NET dataset, start by opening the copy you made of Reporting_App at the end of the section *An ADO .NET Dataset*. Select Project → Add New Item, and select Crystal Report from the list of available templates. Call the report adonet_sample.rpt, click Open, and the familiar Crystal Report Gallery will open with options for creating your report.

Again, we are going to use the Standard Report Expert, although you could use any other expert you wish to get started. Because we are using the Standard Report Expert, the first step is to select the data for our report. The Data Explorer contains a folder for adding project data.

All of the dataset objects that were generated during the build will be listed here, in the format of Project.datasetName (for example, Reporting_App.CustomerOrders). Simply add the tables from the dataset to your report to gain access to all of the fields within the dataset.

You can then use these tables and fields in your report development just like using any other source. Click the Fields tab to select the fields that will appear in your report. Choose CompanyName, City, and Region from the Customers table. Then select OrderDate and ShippedDate from the Orders table.

Select Finish to create the report from your ADO .NET dataset. Now you can simply save the report and integrate it into your application. The following section outlines how to use this type of report with the Windows Report Viewer, and provides a brief note of how to use it with the Web Report Viewer.

Viewing Reports Containing an ADO .NET Dataset

If you were to set the ReportSource property for the Windows Report Viewer to your newly created report based on ADO .NET, and previewed it exactly the same as you would a normal report, nothing would appear. By default, the ADO .NET dataset we have used does not contain any data.

To continue from our previous example where we created a report from the dataset based on the Customers and Orders tables, we are going to look at the code required to integrate this report (based on an ADO .NET dataset) into a Windows application.

To start, we need a form to host the Crystal Reports Viewer. From your project, open Form1.vb, and drag the CrystalReportViewer from the Toolbox to the form. Next, add the report as you normally would, by dragging ReportDocument from the Components section of the Toolbox, and selecting our new report, adonet_sample.rpt.

Next we have to add an OleDbDataAdapter just as we did in the data viewer project earlier. This involves dragging the control onto our form, and stepping through the Data Adapter Configuration Wizard, which in brief comprises (look back to the “Viewing the Contents of a Dataset” section if you need a guide through these steps):

- ❑ **Choosing your data connection**—If it isn’t the default, enter the details for your Northwind database.
- ❑ **Choosing a query type**—We will use SQL statements.
- ❑ **Generating the SQL statements**—Click the Query Builder button, and use the Add Tables dialog to add the Customers and Orders tables as we did before. Select (All Columns), and check the SQL generated is as follows:

```

SELECT
    Customers.*,
    Orders.*
FROM
    Customers
INNER JOIN
    Orders
ON
    Customers.CustomerID = Orders.CustomerID
  
```

Notice in Figure 7-30 how `OleDbConnection1` has also appeared on the report next to `OleDbDataAdapter1`. This is generated from the connection information that you provided the Data Adapter Configuration Wizard.

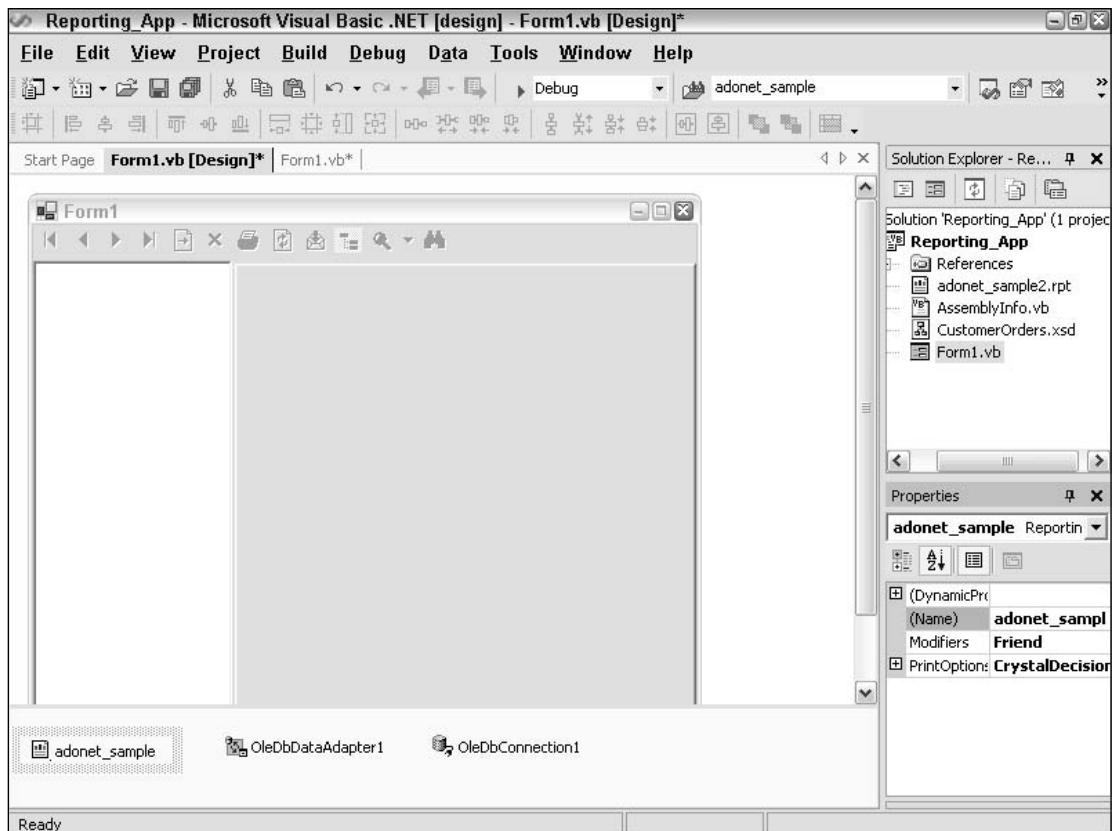


Figure 7-30

With the form now set up to access the dataset, we need to look at some code. Add the same namespaces as we did to view the dataset earlier in this chapter. In addition, because we will be printing a report based on this data, we need to add the `CrystalDecisions.CrystalReports.Engine` namespace to the code behind this form, as shown here:

Chapter 7

```
Imports System.Data
Imports System.Data.OleDb
Imports CrystalDecisions.CrystalReports.Engine

Public Class Form1
Inherits System.Windows.Forms.Form
```

Next we need to add variables to dimension the connection, the data adapter, and the dataset, and get the form ready to accept data. We will connect to the Northwind database using `OleDbConnection1` (as seen on the form designer), and submit an SQL query that will return a result set, filling the `Customers` table, and then the `Orders` table in the dataset. To do this, double-click the `CrystalReportViewer` in the Form Designer to generate the `CrystalReportViewer1_Load` procedure. Then insert the following code:

```
Private Sub CrystalReportViewer1_Load(ByVal sender As System.Object,
                                      ByVal e As System.EventArgs)
    Handles CrystalReportViewer1.Load

    Dim myDataSet = New CustomerOrders()

    Dim sqlString As String = "Select * from Customers"
    OleDbDataAdapter1 = New OleDbDataAdapter(sqlString, OleDbConnection1)
    OleDbDataAdapter1.Fill(myDataSet, "Customers")

    sqlString = "Select * from Orders"
    OleDbDataAdapter1 = New OleDbDataAdapter(sqlString, OleDbConnection1)
    OleDbDataAdapter1.Fill(myDataSet, "Orders")

End Sub
```

With the data now in the dataset, we can set our report source and preview the report itself, by adding this code to the bottom of the `CrystalReportViewer1_Load` procedure:

```
OleDbDataAdapter1.Fill(myDataSet, "Orders")

Dim myReport = New adonet_sample()
myReport.SetDataSource(myDataSet)
CrystalReportViewer1.ReportSource = myReport

End Sub
```

When you run your application and view this form, your report will be filled with data from the ADO .NET dataset, and will be displayed within the viewer.

The only difference in integration between the Windows and the Web Crystal Report Viewer is that you will need to add the appropriate report viewer for the environment you are working with (discussed in Chapter 4, “Report Integration for Windows-Based Applications,” and Chapter 5, “Report Integration for Web-Based Applications”). You can change the contents of the ADO .NET dataset numerous times within your application, and you can call the `DataSource` method at any time before previewing or refreshing your report.

Summary

In this chapter, we had a look at how Crystal Reports .NET accesses different datasources, as well as looking at some of the options for working with data within the Report Designer itself. We also had a brief look at creating ADO .NET datasets and walked through how to design and integrate reports using them.

With all of this behind us, it is time to look at one of the most time-consuming tasks when creating reports—integrating formulas and logic in our report, which is covered in Chapter 8, “Formulas and Logic.”

8

Formulas and Logic

It is very seldom that you will create a report that is just a simple list of information. Most reports will include some sort of calculation or logic for everything from a simple calculation to complex statistical work to manipulation of strings, dates, and so on. In addition to having its own formula language, Crystal Reports can also be used to create reports that take advantage of logic and calculations performed elsewhere as well, including database views, stored procedures, and SQL commands and expressions.

In this chapter, we are going to be looking at all of these methods of integrating calculations into your report and will spend time learning about where the majority of Crystal Reports development time is spent: writing formulas and logic. This will include:

- Deciding how to integrate logic into your report
- Working with the Formula Editor
- Creating formulas with Basic syntax
- Creating formulas with Crystal syntax
- Writing record selection formulas
- Using conditional formatting

At the end of this chapter, you will be able to identify the best way to add calculations and logic to your report and understand enough syntax and code to handle most situations. You should also be able to differentiate between the two different flavors of the Crystal Formula Language and to write your own record selection and conditional formatting formulas.

Integrating Formulas and Logic into Your Reports

Few reports are ever created just to list information on a page. Most will include at least some basic calculations, summaries, and logic. You wouldn't want an invoice, for example, that didn't have a sales tax calculation and total at the bottom or a sales summary without any totals. How this logic is incorporated into your report depends on the requirements set out by the eventual user. The following sections give an overview of the different ways that you can incorporate formulas and logic into your report.

The examples in this chapter deal specifically with creating and working with formulas. If you decide that you need to use a method other than writing formulas, turn back to Chapter 7, "Working with .NET Data," for examples of how to work with application data, SQL Commands, and SQL Expressions.

Database Structures

Since all reports are based on a data source and the most common type of data source is a database, it follows that you might place the logic for your report in the database structures that you're using. In addition to creating reports from database tables, Crystal Reports .NET can also grab data from *views* and *stored procedures*. Using these devices, you can push most of the processing back to the database server. The advantage of doing this is that you can make further use of the investment you've made in your database server. All of that processing power will translate to reports that run faster and more efficiently.

In instances in which you need to reuse logic and particular representations of the data between different reports, views and stored procedures can provide a definite advantage over the raw data structures:

- ❑ Using a view, you can perform joins and consolidation at the level of the database so that, when you design your report, there are no messy joins to configure, and you don't have to figure out how the pieces fit back together. This is especially handy if you have a complex set of data structures or if you foresee end users creating their own reports some day.
- ❑ If you have calculations or business logic that gets used repeatedly, a stored procedure can provide those calculations for a number of reports without having to cut and paste Crystal formulas between the reports themselves. This is sometimes called *2½-tier logic*, rather than a true *3/n-tier* situation, but the benefits are still tangible: The business logic can still be broken out and reused in other reports. You can also add parameters to your stored procedures, and Crystal Reports will accept input to these parameters as if they were a native Crystal Parameter field.

There will be times when you want a very specific report from multiple data sources, but you find it difficult to select and display the data you need using Crystal Report's native features and formulas. For example, you may have data from three or four disparate order entry systems and need to show information from all of them in one report. The data itself is stored in different formats and tables with different primary keys and structure.

You could spend time creating Crystal Reports from these different data sources and then attempt to use sub-reports to join the data together. This method would allow you to create composite keys to pass variables back and forth between the different sub-reports and to try to massage the information into the format you need. Alternatively, you could write a stored procedure to consolidate the data and use that as the data source for your report.

For example, you could work with various Crystal features to create a report that would show data from the New York Stock Exchange alongside world economic indicators from the UN and your own company's sales—but what would happen if you wanted to add another data source or additional features to your report?

A key component of the report development lifecycle that we talked about in Chapter 2, “Getting Started with Crystal Reports .NET,” was the *technical review*. This is where you review the report’s planned content and determine the best way to deliver it. If the best method is to use a stored procedure or a view, don’t be afraid to use the tools you have at hand. In an example with three data sources, it would probably make sense to consolidate all of these sources into one database and then develop a view or a stored procedure to put them together in a format that could be used to create the required output.

Invariably, using stored procedures or views to return a result set will cause someone to ask, “If we’re writing all these stored procedures, why do we need Crystal Reports at all?” We need Crystal Reports to do what it does best: creating information-rich, presentation-quality reports from the data provided. The better the dataset that we provide to Crystal Reports, the easier report development will be, and the more features we can incorporate into our report.

Application Data

Most applications today are developed on an underlying database or other data source, and Crystal Reports .NET can use this application data as the source for your reports, as we saw in the last chapter.

Probably the most compelling reason to use application data is that the logic from the application can be reused in your report. For example, if you have a data-bound grid that displays a number of orders together with calculated fields for order totals, sales tax, shipping, and so on, you could reuse this dataset as the source of your report.

If you didn’t have access to this data, you would have to re-create the calculated fields using Crystal Reports formulas. When something changed in the application, it would also have to change in each report where it appeared. If you’re developing an information-rich application in which the business logic is an integral part of creating that information, you may want to consider using the application data for the logic or calculations that appear in your report.

Crystal SQL Commands

As we saw in Chapter 6, “Creating XML Report Web Services,” something new to this version of Crystal Reports is the ability to use SQL commands as the basis of your report. In previous versions, all of the tables, joins, groups, fields, and so on that you selected for your report were translated into a SQL statement that was written by Crystal Reports itself. That request was submitted to the database, and the results returned to the Report Designer for formatting.

The problem was that, while you could view the SQL statement generated within the Report Designer, you couldn’t really change it or use an existing statement in its place. Setting the SQL statement at run time provided one way of dealing with this limitation, but you still had to rely on Crystal Reports to create the initial SQL statement based on the tables, joins, and so on that you selected in the report’s design.

Clearly, there had to be a better way of doing things. With this release of Crystal Reports, you can now use SQL commands as the data source for your reports, as shown in Figure 8-1, as opposed to tables, views, stored procedures, and so on.

Chapter 8

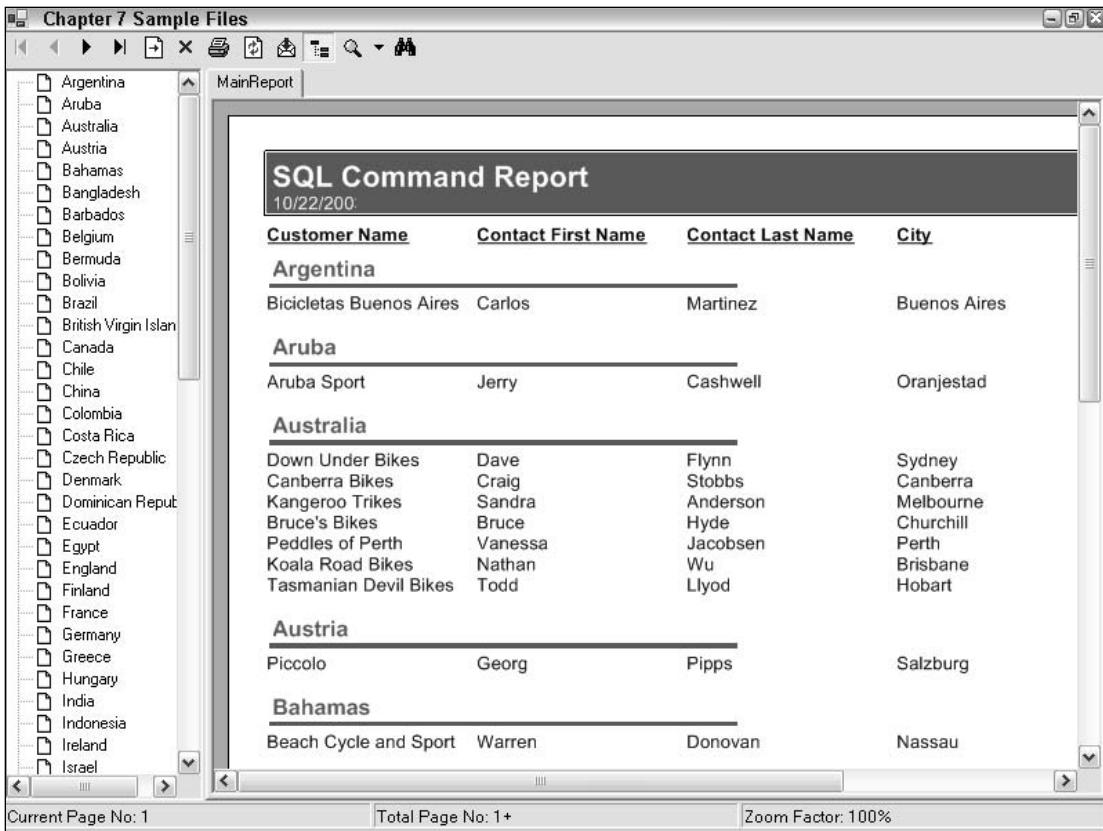


Figure 8-1

The only drawback to creating a “virtual table” with an SQL command is that, when the database changes (when changing a field type, for example), you will need to go back to the report and change the SQL statement to reflect it.

Also, SQL commands cannot be shared between reports so you could end up cutting and pasting the SQL statement between them. If you’re considering implementing a report using a SQL command, you may want to consider creating a *database view* using that same SQL statement. In addition to being able to be used by multiple reports, a view is probably easier to maintain in the long run than individual SQL statements held within Crystal Reports.

Crystal SQL Expressions

Another key database feature of Crystal Reports is the ability to create SQL expressions using the purpose-built SQL Expression Editor, as shown in Figure 8-2:

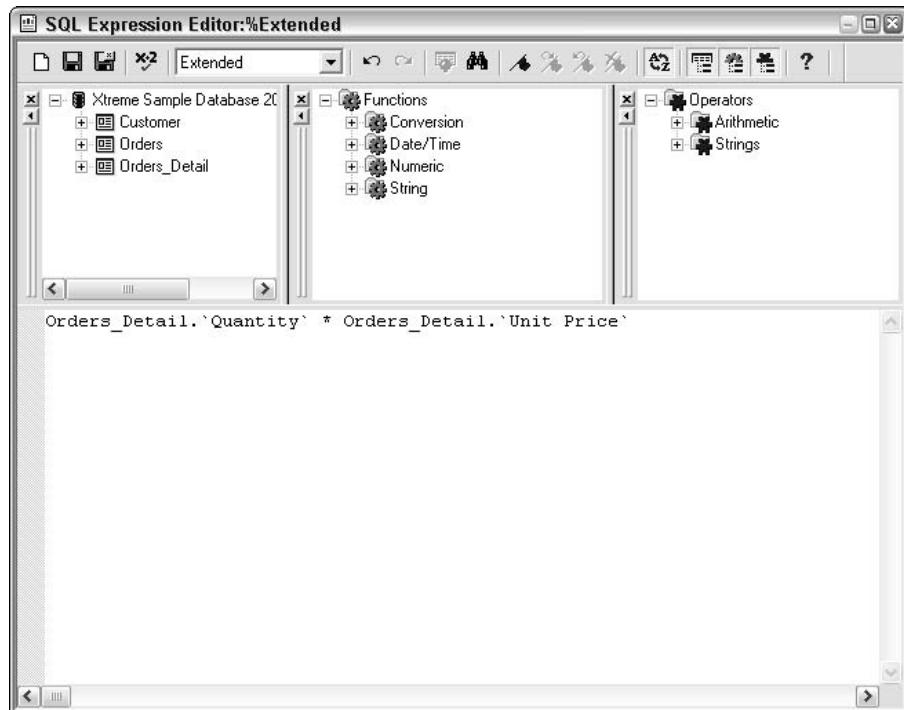


Figure 8-2

When creating a SQL expression field, you have access to all of the SQL functions and operators that are supported by your particular database server. These expressions are evaluated directly on the server with their values returned to Crystal Reports. Be aware, though, that the same function or feature may not be available if you change the source of the report from one database platform to another.

If you're planning to use SQL expressions, you may want to turn back to Chapter 6, "Creating XML Report Web Services," to review best practice for using this feature.

SQL expressions are a handy way to add discrete calculations to a report, but, if you find yourself creating the same expressions for multiple reports, you may want to consider using a stored procedure, a view, or a SQL command to eliminate repetitive coding.

Formulas

Crystal Reports .NET has its own feature-rich formula language for adding calculations and logic to your report. Where the methods listed previously can offer advantages in terms of sharing or reusing logic or employing the power of a database server, Crystal's formula language has advantages of its own.

To start with, this language is fully integrated with all of the features in your report. We could use a field in a stored procedure to do a calculation such as computing sales tax, but it is with the Crystal Reports formula language that we can conditionally format that field or print out text when the sales tax value is over \$10,000, as shown in Figure 8-3:

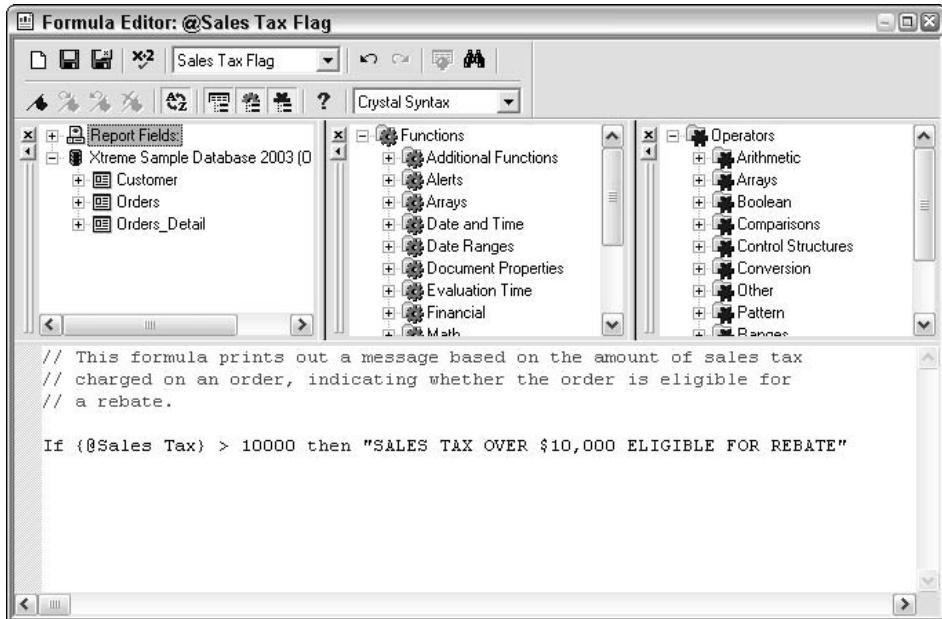


Figure 8-3

When working with a formula within Crystal Reports, you have access to the report’s “print state” and document properties, which include the page number, summary information, and so on. You could create a formula to print out a message showing the title, author, file name, path, and print date on the last page of the report. This information is not available from any SQL data source—just the report itself.

Another example of when a formula is preferable to working with SQL is when you need to use some of the functions that are inherent to Crystal Reports .NET, like the ones for periods of time (`MonthToDate`, `YearToDate`, and so on), financial ratios (like current ratio), or A/R (Accounts Receivable) turnover. That would be difficult to create from scratch in SQL.

When choosing a method for integrating calculations or logic into your report, you’re likely to use a combination of the methods listed previously, based on their own merits. A good guideline to follow is that, for complex calculations and data manipulation, you’ll probably want to use your database server to its fullest capacity and create summary tables, views, and/or stored procedures to provide the information you need. If you’re working with an information-rich application that already contains some logic and performs calculations on a .NET dataset, you’ll probably want to use the existing data in your application. But, for calculations that can’t be sourced from database structures or application data or that are specific to Crystal Reports’ features and functionality, the Crystal Reports formula language can surely hold its own.

Working with the Formula Editor

There are two basic types of formulas. First, there are *formula fields* that you can insert in your report, which are usually enclosed in braces and prefixed by the @ symbol (for example, the {@SalesTax} shown in Figure 8-4. Second, there are formulas that appear behind the scenes, like those for record selection or conditional formatting. From these two distinct types, you can create thousands of different formulas, but, regardless of what you're working with, formulas are all created, debugged, and edited using the Crystal Reports Formula Editor.

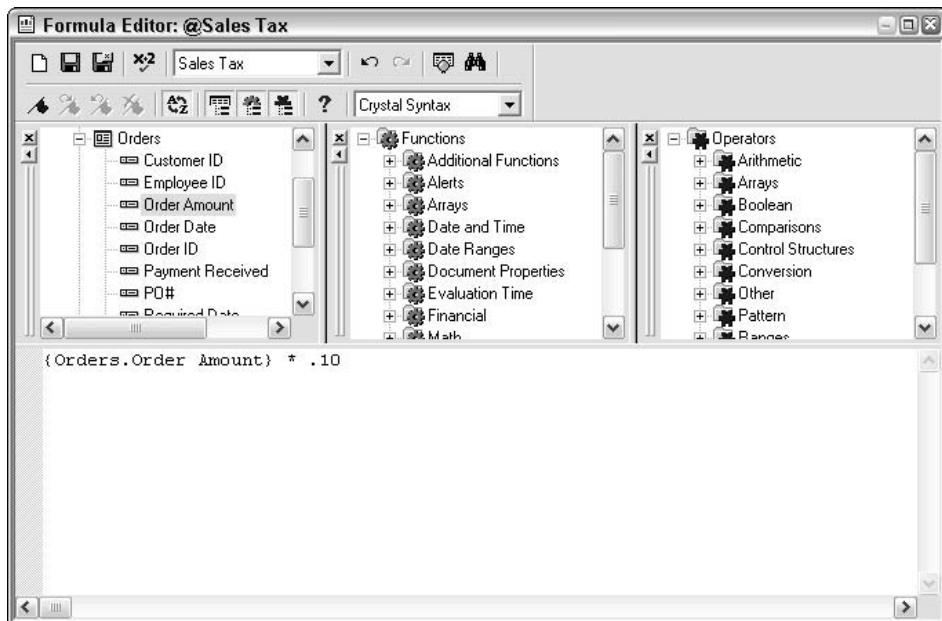


Figure 8-4

Controlling the Editor's Appearance

The Crystal Reports Formula Editor has undergone a number of changes over the past few releases to move from a simple textbox to something that resembles a real code editor, including a customizable interface, color coding, and search and replace features.

To get things underway, the Formula Editor can be opened by either creating a new formula or editing an existing formula. In this example, we're going to look at the Formula Editor by editing a formula that appears in a report (operators.rpt) that is included with the sample files for this chapter. You can open the project that is included with the sample files, or you can create a new project if you want. The main focus of this section is working with formulas that appear in a report.

Chapter 8

Open the report in the Report Designer, and expand the Formula Fields section of the Field Explorer, as shown in Figure 8-5.



Figure 8-5

Locate the Sales Tax formula, directly right-click it, and select Edit. This will open the Crystal Reports Formula Editor, the appearance of which is controlled by a set of default values.

These values can be set in the Report Designer by right-clicking the report and selecting Designer → Default Settings → Editors, as shown in Figure 8-6. You can set the properties for comments, keywords, text, and selected text using this dialog, including changes to the font size and color. Using these properties, the Formula Editor can be modified to your preferences, for example, highlighting comments in a bright color or keywords in bold.

There's also a Reset All button that you can use if you'd like to reset the editor's settings to their original defaults.

Controlling the Syntax Type

When working with the Crystal Reports formula language, there are two different types of syntax available to you: Basic syntax and Crystal syntax. We'll look at each of these in more detail later in the chapter.

Which syntax you are working with is controlled by a drop-down list that appears in the upper right-hand corner of the formula editor. Each type of syntax has its own operators and functions that may (or may not!) overlap.

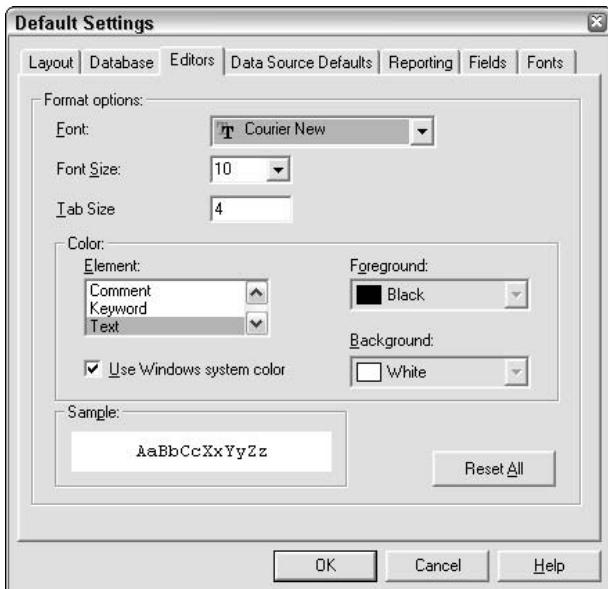


Figure 8-6

Checking for Syntax Errors

To check your formula for syntax errors, there is a Check icon (labeled X+2) that appears on the Formula Editor toolbar. This performs a syntax check on the formula in the window, but it doesn't guarantee that your formula will run or produce the desired result. It just checks to make sure that you've spelled everything correctly and your code is well formed.

With the operators report open in the form designer, right-click the Total formula under Formula Fields in the Field Explorer to open the Formula Editor by selecting Edit from the right-click menu. Within the formula editor, click the Check icon. You will then receive the message, as shown in Figure 8-7:



Figure 8-7

Chapter 8

Click OK, and go back to the Formula Editor. Enter some random characters after the formula, and click the Check icon again. This time, an error dialog is displayed, as shown in Figure 8-8:

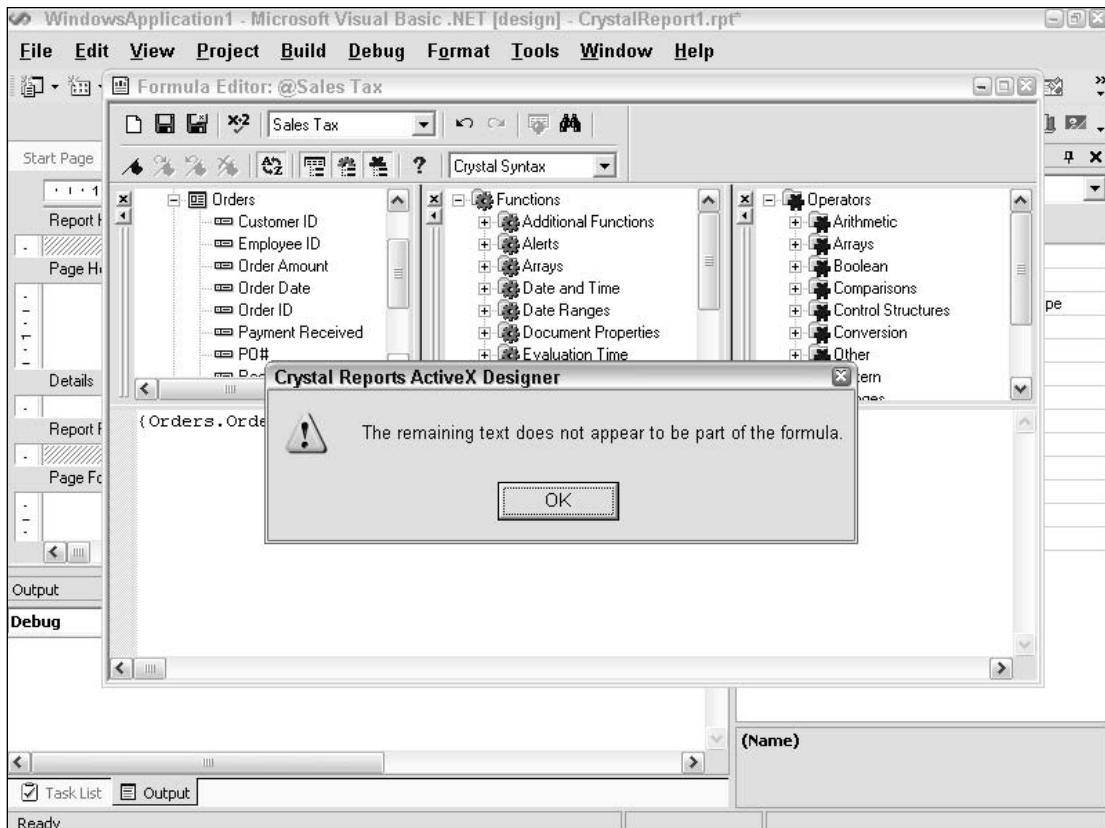


Figure 8-8

In this case, the checker has correctly identified an error in which some characters have been inserted at the end of a formula. If you do not correct this error, you will receive another warning when you attempt to save the formula, as shown in Figure 8-9:

You can select No to leave the formula without correcting the error, but doing so may cause further errors when your report is run.

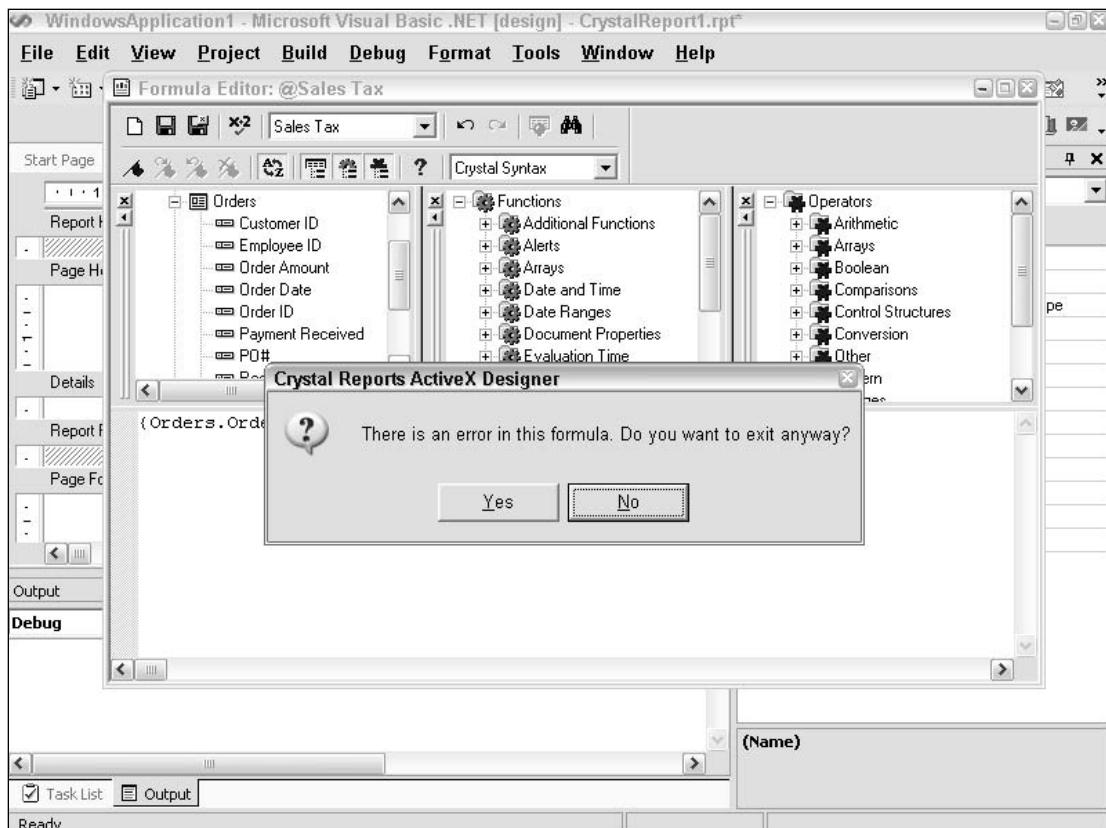


Figure 8-9

Creating Formulas with Basic Syntax

As mentioned above, Crystal Reports has two different types of formula syntax available for use. Crystal Syntax was originally the only syntax available for use in formulas, but Visual Basic developers complained bitterly about having to learn yet another language — particularly one that seemed to be part-Pascal and part-Basic.

What Is Basic Syntax?

With the introduction of Crystal Reports 7.0 came Basic syntax. This closely resembled Visual Basic code by using similar functions and operators but with the ability to access all of the Crystal-specific functions and features. Over time, the two syntaxes have grown closer together with the Crystal version having gone farther to reach its Basic cousin.

Chapter 8

Which syntax you choose depends upon your background and experience. If you're a dyed-in-the-wool Crystal Reports developer, the chances are that you'll be more familiar with Crystal syntax. If you're a Visual Basic developer who has been pushed into report development as well, you'll be more comfortable using Basic syntax.

Since the two versions of syntax have grown closer together, we're going to concentrate our discussion here around the Basic version. A little later in the chapter, we'll look at the differences between the two, which should allow you to use the syntax of your choice.

Basic Syntax Coding Conventions

The structure used by Basic syntax in Crystal Reports closely resembles the structure used in Visual Basic, but there are a few slight differences. Open `operators.rpt` from the sample code, right-click Formula Fields in the Field Explorer, and select New. Enter "LearnSyntax" in the dialog as the name of the formula.

To start with, field names are enclosed in braces and use the naming convention of `{tablename.fieldname}` so enter the following formula that would calculate extended price in `operators.rpt`:

```
{Orders_Detail.Quantity} * {Orders_Detail.Unit Price}
```

Make sure that the Crystal syntax is selected, not Basic syntax, and click the Check icon. The formula should be fine.

Other fields use a prefix to indicate the type of field you're working with. Parameter fields, for example, are prefixed by a question mark, and formula fields are prefixed with the @ character. Change your formula to calculate the sales tax using the extended price from this report by entering the following:

```
{@Extended Price} * .10
```

Click the Check icon again to confirm that the formula is correct since we are writing this formula in Crystal syntax. Now that you can drag and drop the formula field onto the Details section of a report, it would be evaluated once for each record, and the value would be displayed.

However, Basic syntax is slightly different. For each formula you write in Basic syntax, you need to use a special `Formula` variable to tell Crystal Reports what to return. Using the same `LearnSyntax` example in the Formula Editor, change the syntax to Basic syntax using the drop-down menu, and click the Check icon.

The correct code for our sales tax formula would be:

```
Formula = {Orders.Order Amount} * {?SalesTax}
```

Enter this formula, and click the Check icon to ensure this line of code is correct. Even if you don't need to output a value, you still have to use the `Formula` variable and assign some value to it (even if you just make it up).

If, for example, we create a global variable and insert a calculation to add up the number of orders as we go down the page, but we don't actually want to print anything out until the end, we still have to set the Formula variable to avoid getting a syntax error. Try the checker on the following code with and without the line that says `Formula = 999`.

```
Global TotalOrders as Number  
TotalOrders = TotalOrders + {Orders_Detail.Quantity}  
Formula = 999
```

As you've probably noticed, assignments are made using the equal operator. Just like other versions of the Basic language, you can add comments to your formulas with either the single quote or the `REM` statement:

```
' This formula calculates the Total Sales
```

```
Global TotalOrders as Number  
TotalOrders = TotalOrders + {Orders_Detail.Quantity}  
Formula = 999
```

```
REM The formula variable is required
```

If you want to use `REM` on the same line as some formula text, you need to add a colon before you begin your `REM` statement, as shown in the following code:

```
Formula = 999 : REM The formula variable is required
```

If you're using the apostrophe, you can just append it to the end of the line:

```
Formula = 999 ' The formula variable is required
```

Simple Operators

Now we need to look at a few of the simple operators that are available for use. Some of these are used in the sample reports that are included with the download files for this chapter. Many are self-explanatory and don't need much guidance for use so we won't describe how to use every one. The easiest way to become familiar with the large number of operators is to actually use them or play about with them in the Formula Editor using the Check button to ensure your syntax is correct.

Although the majority is the same, Basic syntax and Crystal syntax occasionally utilize different operators. These differences are explained in Appendix C, Crystal vs. Basic Syntax.

Arithmetic

Crystal Reports .NET supports all of the basic arithmetic operators, including addition, subtraction, multiplication, and division, but it also has support for a number of others, as discussed in the following table:

Chapter 8

Operator	Symbol	Description
Integer Divide	\	Division in which only the integer is returned (for example, 9\2 would return a result of 4)
Modulus	Mod	For dividing two numbers and returning the remainder
Negate	- ()	For negating or changing the sign of a number
Exponentiate	^	For exponents, used for calculating squares (for example, 3^2 would return 9)

Open operators.rpt in the Report Designer, and create a new formula by right-clicking Formula fields. By using the negate operator, we could calculate a value representing the number of items returned to the company from an order:

```
-({Orders_Detail.Quantity})
```

The negate function is also useful when working with financial information, in which a negative amount may indicate a credit.

Boolean

For formulas and logic that need to return a True or False value, we also have a number of Boolean operators available within Basic syntax.

Operator	Description
Not	Reverses the value (for instance, Not (True) is False)
And	Returns True where all conditions are True and returns False where one condition does not meet the criteria
Or	Returns True if one or the other condition is met or both
Xor	Returns True if one and not the other condition is met
Eqv	Returns True if both values compared are True or if both values compared are False and returns False if the two values compared are different
Imp	Returns True if the first condition is True and the second condition is False (Otherwise returns True)

Comparison

For comparing two values, Basic syntax supports the usual comparison operators, including:

Operator	Symbol
Equal to	=
Not equal	<>
Less than	<
Greater than	>
Less or equal	<=
Greater or equal	>=

Type Conversion

Within Crystal Reports, there are eight different data types available for use:

- Boolean
- Number
- Currency
- Date
- DateTime
- Time
- String
- BLOB (Binary Large Object)

BLOB fields can be inserted into a report, but they cannot be converted to any other field type. They are handy when you need to insert non-traditional records into your report. (The sample database, for example, has a graphic file inserted into the Employee table that, when placed on your report, will display the employee's photo. This can be seen in the `Employee_Listing.rpt` report that's included with the chapter's sample files.)

When working with all of these different types of fields, we sometimes need to perform a conversion before we can use them in our formulas (for example, when a numeric value is stored as a string in the database). To convert field types, we have the conversion functions shown in the following table:

Function	Use
<code>CBool()</code>	Returns <code>True</code> if the argument is positive or negative but not zero, and returns <code>False</code> if the argument is zero
<code>CCur()</code>	Converts Number, Currency, or String types to Currency
<code>CDbl()</code>	Converts Number, Currency, or String types to Number

Table continued on following page

Chapter 8

Function	Use
CStr()	Converts Number, Currency, or Date types to String
CDate()	For converting to a true Date field
CTime()	For converting to a Time field
CDatetime()	For converting to a DateTime field
ToNumber()	For converting String and Boolean types to Number
ToText()	For converting Number, Currency, Date, Time, or Boolean to text
ToWords()	For spelling out numbers or currency values (for example, 101 is "One hundred and one")

In addition to these, there are also functions for converting **DateTime** strings. You'll find them in the Formula Editor, under the heading Additional Functions. These functions will accept a **DateTime** string and return a Date field, a Time field, or a number of seconds.

So, to convert a number to a currency-format field, the formula would look something like this:

```
Formula = CCur({Orders.OrderAmount})
```

or, to use the ToWords() function to spell out the same currency amount:

```
Formula = ToWords(CCur({Orders.OrderAmount}))
```

which, for a value of 1001.50, would return the string One thousand and one and 50/100.

For date functions, you can pass the date in any number of formats, including the full date, days elapsed since 01/01/1900, date literals, and date components:

```
CDate("Sep. 05, 2002")
```

which returns a date value for September 5, 2002,

```
CDate(#Jan. 01, 2005 12:02pm#)
```

which returns a date value for January 1, 2005,

```
CDate(1960, 10, 10)
```

which returns a date value for October 10, 1960.

Summary Functions

When you insert a summary into your report, a corresponding summary function is used to create a specialized summary field. These summary fields can be used within your formulas just like any other field, and include sums, averages, and counts.

Summary fields are generally shown in one of two different ways. The first of these, where the summary function and a single field are shown, represents a grand total that may appear on your report, usually in the report footer. An example of this type would be:

```
Sum({Customer.Sales})
```

In addition to grand totals, summary fields can also be used at the group level. So, in the same report that shows the previous summary, you could also have a summary field that appears in the group header or footer. For example, if the group field were {Customer.Country}, the summary field would look like this:

```
Sum({Customer.Sales}, {Customer.Country})
```

For more information on inserting summary fields into your report, turn back to Chapter 2, “Getting Started with Crystal Reports .NET.”

String Functions

The string functions within Crystal Reports are used to manipulate database and other fields that contain text. When working with strings, we can concatenate two strings together using either the plus operator (+) or the ampersand (&).

When working with two strings, we could use the plus operator, as shown:

```
Formula = "This is the Customer Name " + {Customer.Name}
```

To concatenate a string and another type of field with the plus operator, we would first have to do a type conversion to ensure that both of these fields were strings. Using the ampersand operator, you can concatenate strings with any other type of field without performing a type conversion first:

```
Formula = "This is the Sales Amount " & {Customer.SalesTotal}
```

Although this method is easier, you may still need to perform a type conversion in order to have more control over how the field is converted, such as setting the number of decimal places when moving from a number to a string.

In addition to concatenating strings, you can reference individual characters or sets of characters using the subscript operator, noted by square brackets ({fieldname} [n], in which n is a position within the string).

To return the first letter of a customer’s first name, the formula would look like this:

```
Formula = {Customer.Contact First Name}[1]
```

You’ll notice here that Crystal treats strings as 1-based arrays (instead of 0-based). It also provides the ability to return a range of characters from the array —in this case, the first three letters:

```
Formula = {Customer.Contact First Name}[1 to 3]
```

Chapter 8

In addition to concatenating and pulling strings apart using simple operators, we have a number of functions that can be used with string-type fields, including:

Function	Description
<code>Len(string)</code>	Finds the length of a string
<code>Trim(string)</code>	Trims extra spaces from either side of a string
<code>LTrim(string)</code>	Trims extra spaces from the left side of a string
<code>RTrim(string)</code>	Trims extra spaces from the right side of a string
<code>UCase(string)</code>	Converts a string to all uppercase letters
<code>LCASE(string)</code>	Converts a string to all lowercase letters
<code>StrReverse(string)</code>	Reverses the order of a string
<code>IsNumeric(string)</code>	Tests to see if a field is numeric
<code>InStr(string1, string2)</code>	Searches for the position of <i>string2</i> inside <i>string1</i>
<code>InStr(start, string1, string2)</code>	Searches for the position of <i>string2</i> inside of <i>string1</i> using a numeric starting point

For example, to find the length of a string, you'd use the `Len` function:

```
Formula = Len({Customer.Country})
```

In the sample report (`employee_listing.rpt`), some of these functions have been used to create an e-mail address to be displayed on a report that follows the naming convention of "first initial, last name," combined with the domain name:

Date and Period Functions

Since most reports will involve date and time information in one form or another, Crystal Reports includes a number of predefined periods to help make life a little easier. You can think of the periods as pre-built arrays of dates, based on the current date. For example, you could use the period `LastFullWeek` in a comparison, and the range of date values from the Sunday to Saturday of the previous week would be included.

A complete list of these period functions appears below:

- `WeekToDateFromSun`
- `MonthToDate`
- `YearToDate`
- `Last7Days`
- `Last4WeeksToSun`
- `LastFullWeek`

- LastFullMonth
- AllDatesToToday
- AllDatesToYesterday
- AllDatesFromToday
- AllDatesFromTomorrow
- Aged0To30Days, Aged31To60Days, Aged61To90Days
- Over90Days
- Next30Days, Next31To60Days, Next61To90Days, Next91To365Days
- Calendar1stQtr, Calendar2ndQtr, Calendar3rdQtr, Calendar4thQtr
- Calendar1stHalf, Calendar2ndHalf
- LastYearMTD
- LastYearYTD

Just to drive the point home, all of the above functions act like arrays of dates. In the function AllDatesFromTomorrow, an array is created behind the scenes that include all dates from tomorrow onwards. Likewise, when you access the Aged0To30Days function, an array is built of all dates that are 0 to 30 days behind the date you are comparing with. This is especially handy when working with financial reports, when you need to show aging of debts, invoices, and other time-sensitive documents.

We use these functions by comparing date-type fields against them to determine whether those dates fall within the represented period. For example:

```
If {Orders.OrderDate} In Over90Days Then Formula = "Overdue"
```

This also can be used to create complex month-to-date and year-to-date reports by displaying the data that falls in these periods in two separate columns.

In this chapter, we will look at an example of a report that has been created from the Customer and Orders tables within the Xtreme sample database (`customer_orders.rpt`, which is included with the code download). Some basic fields have been displayed on the report, like Customer Name, Order Date, and Order Amount, with a grouping inserted on Customer Name.

So, open the sample application, `BasicSyntax_basic`. You may prefer to build this example from scratch by creating your own project and viewer. It makes no difference to the finished result since we will be working more or less exclusively in the Report Designer.

To display the two columns, we need to create two separate formulas, which are then summarized, and the details of the report hidden. For the month-to-date column, locate the Formula Field section of the Field Explorer. Right-click, and select New. Enter a name of `MTD`. The formula text looks something like this:

```
If {Orders.Order Date} In MonthToDate Then Formula = {Orders.Order Amount}
```

Chapter 8

For the year-to-date column, repeat the same process, but name the formula YTD. Enter the formula text shown here:

```
If {Orders.Order Date} in YeartoDate Then Formula = {Orders.Order Amount}
```

Having defined your two formula fields, you can drag and drop them onto your report in the Details section, as shown in Figure 8-10.

The next step in creating our summary report is to directly right-click the MTD field and select Insert Subtotal from the context menu. Then, repeat that for the YTD field

To finish off, right-click the Details section in the Report Designer, select Hide Section, and then do the same for the Group Header #1 on Customer Name.

Your report should now show the two columns for month-to-date and year-to-date values. To get the customer name to appear as well, drag the Group #1 Name field out of the header into your Group Footer. Your report should appear, as shown in Figure 8-11.

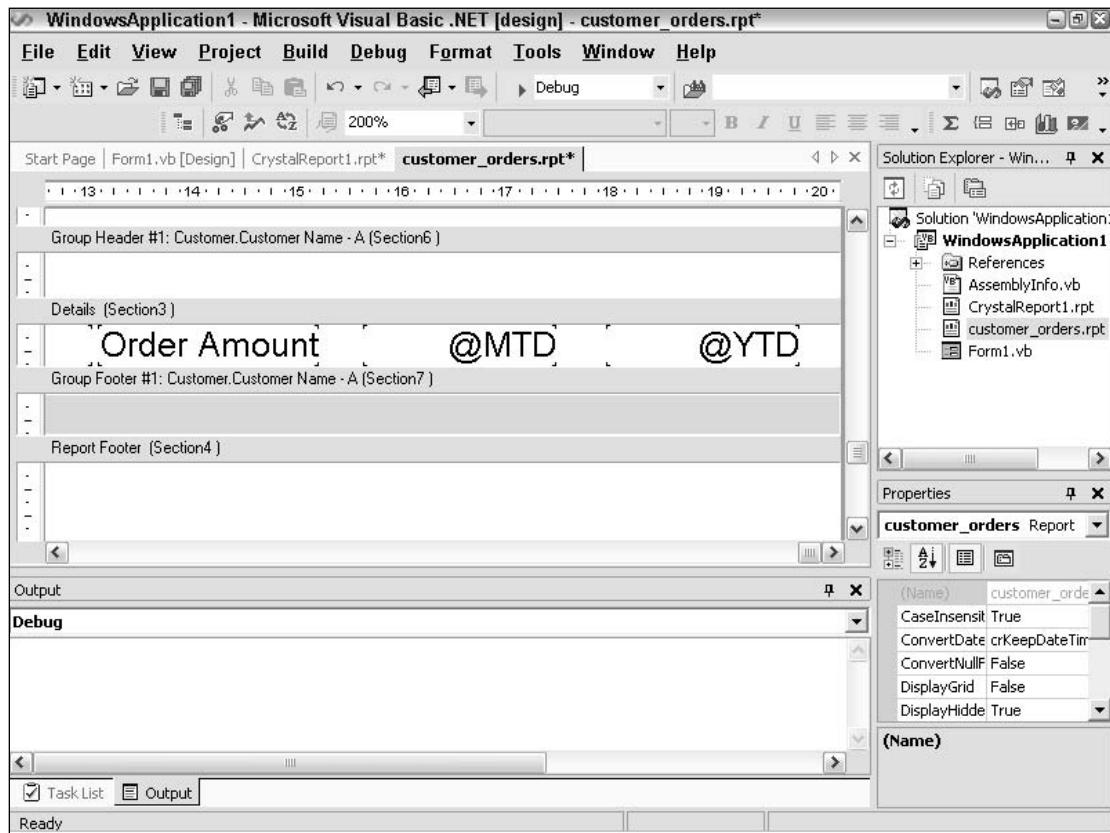


Figure 8-10

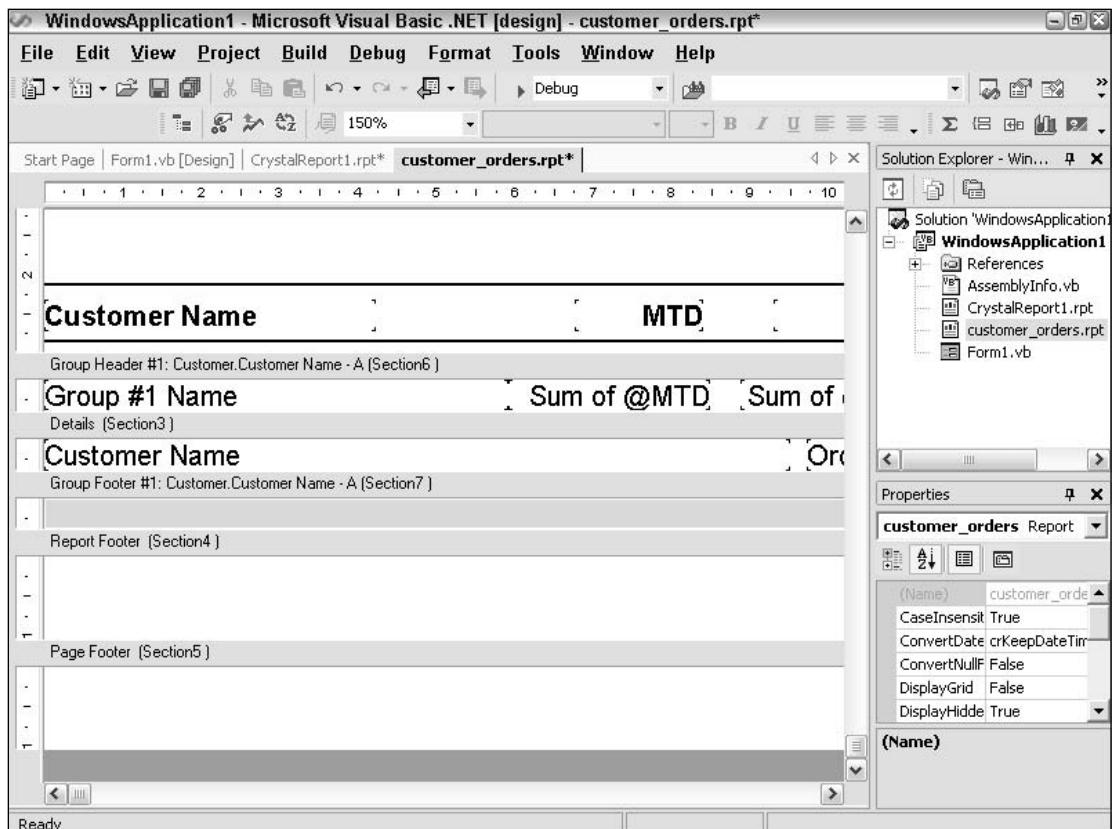


Figure 8-11

If you'd like to have a closer look, the complete report is available in the download file for this chapter.

Compile and run the report—but don't be alarmed if you get a nasty surprise!

Does your report look similar to the download? Probably not—there's one little trick that we forgot. Since the periods we used in this report are built from the current date and evaluated against the sample data (which is at least a couple of years old), the chances are that your MTD and YTD columns will be empty. Happily, you can set the system date in Crystal Reports by right-clicking the report, selecting Report → Set Print Date, and entering a new date and time, as shown subsequently.

This allows you to change the internal date and time used by Crystal Reports to build the time periods and some other date- and time-driven functionality. After setting the date, Crystal Reports will think that it's actually processing the report on that date and behave accordingly. (This is especially handy if you need to do point-in-time reports.)

In our case, if you set this date to 07/01/2001 or 20/12/1997 (either of these dates should be compatible with the sample data), your report should show both the MTD and YTD columns with the totals for each customer.

Chapter 8

Print State and Document Properties

Crystal Reports has a number of special fields and properties that are generated by the system when a report is run. You will have already encountered some of these fields when adding page numbers or summary information such as the report's title, author, and so on.

Here is a list of these types of functions:

Function	Description
PreviousValue (fieldname)	The value of the previous field to appear
NextValue (fieldname)	The value of the next field to appear
IsNull (fieldname)	Tests whether a field is null
PreviousIsNull (fieldname)	Tests whether the previous field value is null
NextIsNull (fieldname)	Tests whether the next field value is null
PageNumber	Returns the current page number
TotalPageCount	Returns the total page count
PageNofM	Returns the current page number of the total page count
RecordNumber	Returns Crystal Reports internal reference of record number
GroupNumber	Returns Crystal Reports internal reference of group number
RecordSelection	Returns the current record selection formula for the report
GroupSelection	Returns the current group selection formula for the report
OnFirstRecord	Returns a True value when the first record is displayed
OnLastRecord	Returns a True value when the last record is displayed

Using these functions, you can quickly create formulas that can mark new records in a sorted list:

```
If Previous({Customer.Customer Name}) <> {Customer.Customer Name} Then Formula =  
"New Customer Starts Here"
```

or, you could print a text message at the end of your report:

```
If OnLastRecord = True Then Formula = " ***** END OF REPORT ***** "
```

Control Structures

Crystal Reports supports a number of control structures that can control branching within a formula.

If...Then Statements

If...Then statements provide an easy method for controlling branching within your formula text. If...Then statements can work on the basis of a single condition, for instance:

```
If {Customer.Country} = "USA" Then Formula = "Local Customer"
```

In the customer sales report example that's included with this chapter (`customer_sales.rpt`), we can create a Formula Field that will assess if the value in the `Country` field is the USA. If it is, a message will be printed showing the customer as a "Local Customer." To do this, you merely open the Field Explorer, right-click Formula Fields → New, and, after giving the field a name (`local_customer_flag`), enter your formula into the editor.

You can also use an `Else` clause for when the condition is not met, for example:

```
If {Customer.Country} = "USA" Then Formula = "Local Customer" Else Formula =
"International"
```

Multi-line `If...Then...Else` statements can also be used, but keep in mind that, once a condition is met, Crystal Reports will not process the rest of the formula text. For example, let's look at this early version of the formula field `sales_performance_flag` in `customer_sales.rpt`:

```
If {Customer.Last Year's Sales} > 30000 Then Formula = "Excellent job!"
Else If {Customer.Last Year's Sales} > 10000 Then Formula = "Fine job!"
Else If {Customer.Last Year's Sales} > 20000 Then Formula = "Great job!"
```

In this formula, if the value passed was 25,000, the formula would immediately stop on the second line (because the condition has been met), giving an incorrect result. In order to have a multi-line `If...Then...Else` formula work correctly, you need to put the conditions in the correct order, like so:

```
If {Customer.Last Year's Sales} > 30000 Then Formula = "Excellent job!"
Else If {Customer.Last Year's Sales} > 20000 Then Formula = "Great job!"
Else If {Customer.Last Year's Sales} > 10000 then Formula = "Fine job!"
```

In addition to multi-line use, you can also use compound `If...Then...Else` statements, nesting two or more statements in one formula, as shown in the following `compound_if_then` formula field from this report:

```
If {Customer.Country} = "USA" Then
  If {Customer.Last Year's Sales} > 30000 Then Formula = "Excellent job!"
    Else If {Customer.Last Year's Sales} > 20000 Then Formula = "Great job!"
      Else If {Customer.Last Year's Sales} > 10000 Then Formula = "Fine job!"
```

Select Statements

Another popular control structure is the `Select` statement, which can be used with `Case` to evaluate a particular condition. If that condition is `True`, control of the formula will go to the formula text for the met condition, as shown in the following code:

```
Select Case {Customer.PriorityNumber}
  Case 1, 2, 3
    Formula = "high priority"
  Case 4
    Formula = "medium priority"
  Case Else
    Formula = "low priority"
End Select
```

Creating Formulas with Crystal Syntax

Over the past few releases, Crystal syntax and Basic syntax have moved closer together through the development of similar functions and operators. In the next section, we'll have a look at some of the remaining differences.

We'll be using a project called `CrystalSyntax_basic` to look at this syntax. This project is available in the download code from www.wrox.com, or you can build it from scratch by following the instructions. If you decide to build it yourself, create a Windows application now and call it `CrystalSyntax_basic`. Add the customer sales report (which should be in the download sample under the path `Crystal.NET2003\Chapter08\customer_sales.rpt`) into the project by right-clicking the project name in Visual Studio .NET and selecting `Add → Add Existing Item...` and then browsing to the report.

We're now in a position to drag and drop a `CrystalReportViewer` onto the Form and also a `ReportDocument` component. When the dialog opens to request which type of `ReportDocument` component you need, select `CrystalSyntax_basic.customer_sales`.

Insert the following code into the `Form_Load` event:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles MyBase.Load  
    Dim myReport As New customer_sales()  
    CrystalReportViewer1.ReportSource = myReport  
    myReport.Load()  
End Sub
```

Now, compile and run it to make sure the report runs successfully and you can preview the report.

We are now ready to go and look at the syntax behind this report.

Differences from Basic Syntax

To start with, a Crystal syntax formula does not require the `Formula =` tag at the end to output the formula results. This is the statement from the `Local_Customer_Flag` formula field in the customer sales report. You can access this field by opening the report in Visual Studio .NET, selecting `View → Other Windows → Document Outline` from the main toolbar, and, in the tree that appears, selecting `Formula Fields`, right-clicking `local_customer_flag`, and then selecting `Edit`.

Following is a Basic syntax statement:

```
If {Customer.Country} = "USA" Then Formula = "Local Customer" else Formula =  
    "International"
```

By default, a Crystal syntax formula will display the last value that is calculated within the formula (that is, on the last line evaluated), as shown in the following code statement:

```
If {Customer.Country} = "USA" Then "Local Customer" else "International"
```

Also, since Crystal syntax was created before Basic syntax, there were a number of Crystal Reports functions that were actually reserved words within Visual Basic. These functions have been given alternative

names, which means that a function in Crystal syntax and one in Basic syntax can now be named differently from each other.

A list of common functions that are different between the two syntaxes is available in Appendix C, “Crystal VS. Basic Syntax.”

Creating Record Selection Formulas

Record selection is a key component of most reports, for it is here that the results are filtered to show only the information required. In Chapter 2, “Getting Started with Crystal Reports .NET,” we had a first look at simple record selection and some of the operators that can be used to filter your report. Now that we’re working with Crystal syntax, we can look at writing these formulas ourselves.

First, all record selection formulas have to be written in Crystal syntax. There is no option to use Basic syntax. This stems from the fact that Crystal Reports will take the record selection formula you create and write a SQL statement to retrieve the report results. Historically, Crystal syntax was the first type of syntax available, and most work has been done translating it to SQL. You may see Basic syntax supported in future releases, but that’s up to Crystal Decisions!

To access or create record selection formulas, right-click your report and select Report → Edit Selection Formula → Records. This should open up standard Formula Editor Dialog. Note that the drop-down box that enables you to switch between Crystal and Basic syntax is grayed out this time.

When working with record selection formulas, we use the same Formula Editor that you’d use to edit formulas that appear on your report with one difference: a record selection formula will never have any type of output. The sole function of the record selection formula is to apply some condition against fields that appear in your report (where the condition is `True`, the record is returned), as shown in the following examples:

```
{Customer.Country} = "USA" And {Customer.Region} = "NC"
```

If you enter the above formula into the dialog shown previously (running a syntax check first of all by clicking the X+2 icon) and then save and run the application, your report should now be filtered to only show those customers that are in the USA and, specifically, within NC (North Carolina).

When working with a record selection formula, you have access to all of the fields, functions, and operators that are available within the formula language itself, including time periods, for instance:

```
{Orders.OrderDate} In MonthToDate
```

Best practice for record selection suggests that you only perform it on fields originating from your database. For example, imagine you have a formula in your report that translates the state or region names within the database to their “long names” (so “CA” is shown as “California”). If you were going to create a record selection formula that used the state, you would definitely not want to use the “long name” formula. If you did, the generated SQL statement would pull back every single record, evaluate the formula, and then use Crystal Reports’ own inherent selection routines to filter the data.

On the other hand, if you were to create the record selection formula directly from the database fields, this in turn would be written to the SQL statement, and your report will run much more quickly.

Working with Conditional Formatting

Conditional formatting with Crystal Reports provides an effective method for highlighting data within the report. As you moved through the Report Designer, you may have noticed that a number of formatting options had the X+2 icon shown beside them.

This indicates that these formatting properties can be used with conditional formatting: when some condition is True, the formatting property will be applied.

Understanding Conditional Formatting

There are two different kinds of conditional formatting available within Crystal Reports, based on the two different kinds of properties available for Crystal Reports elements. The first deals with Boolean properties, which include things like Suppress and Can Grow, and have an on/off state. The second deals with properties that can have multiple outcomes, like font or section colors. While the concepts behind using these two different types of properties are similar, the formulas behind them are different.

Conditional Formatting for Boolean Properties

With these properties, there is either a True or False status. If a field has the Suppress property checked, the property is True, and it will be suppressed. If not, the property is False, and the field is shown. When working with conditional formatting on these types of properties, all we need to do is specify a condition. If that condition is True, then the property will be set to True.

For the following walkthrough of conditional formatting, open the Customer Sales report (`customer_sales.rpt`) that's included with the sample files for this chapter.

In the case of Suppress, you can right-click a field. In this case, the Last Year's Sales field (not the field in the report header, but the one in the report proper!). Select Format. On the Common property page, you will see the Suppress property checkbox. To start with, don't check the box—that would turn suppression on for every value. Instead, click the X+2 box that appears to the right of the property, as shown in Figure 8-12, to open the Formula Editor, and enter a conditional formatting formula.

Since this is a Boolean-type property, all we need to do is enter a condition. When the condition is True, the formatting option will occur (in this case, the field will be suppressed). The formula would look like this:

```
{Customer.Last Year's Sales} < 10000
```

This would cause our report to show only the values that were greater than 10,000.

This conditional formatting formula was applied to an individual field so only the field is suppressed, but you could apply it to an entire section. Be warned, though, that, when you suppress a field or a section, it only suppresses the display of the field. It will still be included in any totals or formulas.

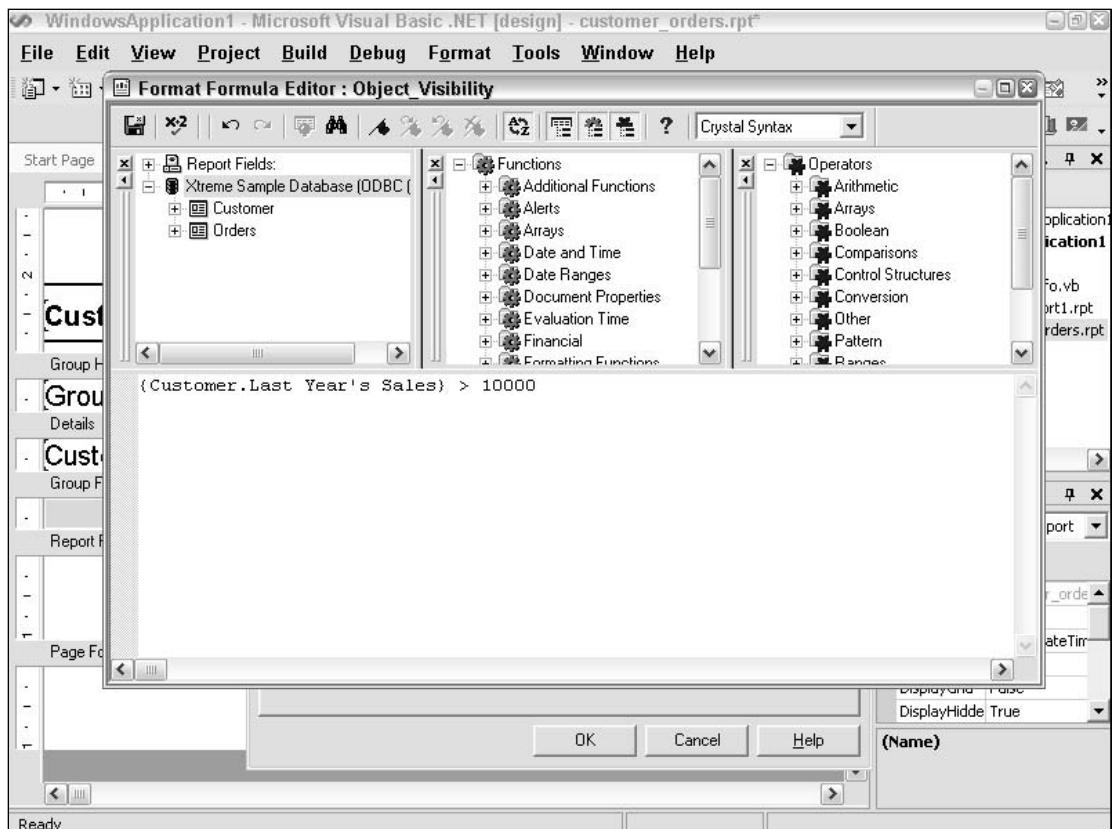


Figure 8-12

Conditional Formatting for Multiple-Outcome Properties

With properties that can have a multiple outcome, the conditional formatting formula needs to be a bit different. In our previous Boolean example, only two things could happen: either the field was suppressed or it wasn't. With multiple-outcome formulas dependent on the setting of a property, any number of things could happen. Using the same condition as above, if our value was over 10,000, we could have changed the field change color—but to which color?

When working with multiple-outcome properties, we have to use an `If . . . Then` statement to specify what will happen when the condition is True. For example:

```
If {Customer.Last Year's Sales} > 20000 Then Red
```

To apply this conditional formatting, again, right-click the Last Year's Sales field, select Format, switch to the property tab for Font, and click the X+2 icon beside the font color. This will open the Formula Editor and allow you to enter your conditional formatting formula, as shown in Figure 8-13.

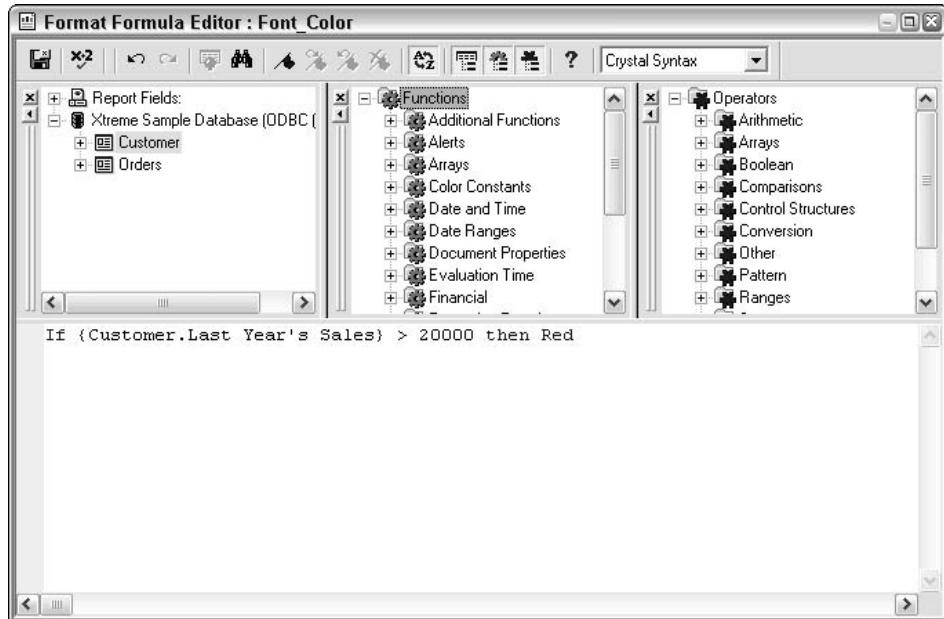


Figure 8-13

In the Formula Editor, all of the available values or constants for a particular property will appear in the function tree, as shown in Figure 8-14.



Figure 8-14

In addition to the constants (with a specific set for each attribute), you can also set the property to its `CurrentFieldValue` or to its `DefaultAttribute`. For example, if you had a field in which the color property had already been set to Green, you could use this default in your formula:

```
If {Customer.Last Year's Sales} > 10000 Then Purple Else DefaultAttribute
```

With conditional formatting on multiple-outcome properties, you are not confined to a single `If...Then` statement. You can also use a compound `If...Then` or an `If...Then...Else` statement, as shown here:

```
If {Customer.Last Year's Sales} > 100000 Then Blue  
Else If {Customer.Last Year's Sales} > 50000 Then Teal  
Else If {Customer.Last Year's Sales} > 20000 Then Red
```

Remember that the same rules apply to these formulas as to the formulas you create to display on your report. Once the condition has been met, Crystal Reports will stop evaluating the rest of the formula.

Summary

In this chapter, we've had a look at integrating formulas and logic into your report and explored the topics of creating formulas to appear on your report, setting record selection, and controlling conditional formatting. In the next chapter, we'll be looking at the Crystal Reports Engine, which is a selection of classes that can help give us much more power and control over our reports and their presentation.

9

Working with the Crystal Reports Engine

Earlier in the book, in the chapters about integrating reports for Windows and Web-based applications (Chapter 4, “Report Integration for Windows-Based Applications” and Chapter 5, “Report Integration for Web-Based Applications”), we looked at the different object models that were available and, in turn, covered both of the object models for working with Windows and Web-based applications.

You may remember that there is actually a third object model that can be used that provides almost full control over the report itself and its contents. This object model, contained within the `CrystalDecisions.CrystalReports.Engine` namespace, is used in conjunction with the viewer object models and provides a powerful method to customize your report at run time.

In this chapter, we will be looking at the Crystal Reports engine, the functionality it provides, and some of the advanced integration techniques that you can use in your own application.

These will include:

- Crystal Reports Engine overview
- Printing and exporting
- Working with databases
- Formatting areas, sections, and fields
- Working with parameter fields
- Customizing formulas
- Working with other report components

At the end of this chapter, you will be able to identify when to use the Crystal Reports Engine namespace, know how to integrate it into your application, and understand how the features contained within can be used to customize reports within your application by using the properties, methods, and events associated with the engine.

Obtaining the Sample Files

All the example reports and code used in this chapter are available for download. The download file can be obtained from www.wrox.com.

You can use this solution to walk through the code examples in this chapter, or you can create your own solution by opening Visual Studio .NET and selecting File → New → Blank Solution. You can then use this solution to follow along with the instructions.

Understanding the `CrystalDecisions.CrystalReports.Engine` Namespace

If you have worked with Visual Basic and Crystal Reports before, you are probably familiar with the functionality within the Crystal Reports Engine. The Crystal Reports Engine provides a low-level interface into the report itself and allows developers to control most aspects of the report programmatically.

Within Visual Studio .NET, the `CrystalDecisions.CrystalReports.Engine` namespace is used to expose this functionality. If you are integrating your reports in a simple, view-only application, you may never need to delve into the Crystal Reports Engine, but, for more complex integration and applications, you will find you'll need to use it almost every time.

Remember our earlier discussion in Chapters 4, “Report Integration for Windows-Based Applications,” and 5, “Report Integration for Web-Based Applications,” about which namespace you should use for what function? One of the key points of that discussion was that you shouldn’t mix the object models in your code. For example, if you do choose to use the `CrystalDecisions.CrystalReports.Engine` namespace, set all of your properties, methods, and events relating to the report using this namespace, and use the `CrystalDecisions.Windows.Forms` namespace only for setting your report source, viewing the report, and modifying viewer properties (such as toolbars and icons). Never the twain shall meet!

Customizing Reports Using the Report Engine

To get started, we will need to create a new project within our solution. To create a new project, select File → New → Project. In this instance, create a Windows application, and call the project `engine_basic` because, in the following sections, we are going to be looking at some basic Report Engine functionality.

We need a report object to work with. To add the sample reports to your project, select Project → Add Existing Item, and select the folder where you unzipped the sample project. You will also need to change

the file extension from VB Code Files to All files to see the report files we will be using in this chapter, including the first report, `Crystal.NET2003\Chapter09\employee_listing.rpt`.

With a report to work with, we now need to reference the Report Engine so we can use it in our application. This reference may well add itself when you add the report to the project, but, if not or if you are working with one of your custom reports, this will have to be added in Visual Studio .NET. To add a reference, select Project → Add Reference to open the dialog shown in Figure 9-1:

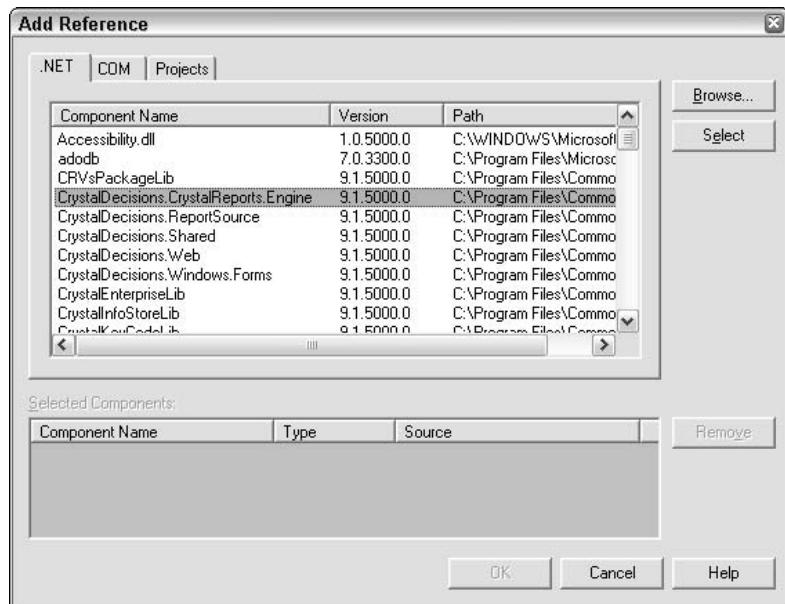


Figure 9-1

Highlight the `CrystalDecisions.CrystalReports.Engine` namespace, and click the Select button and then the OK button to add the reference. The reference to the Crystal Report Engine should now appear in your project in the Solutions Explorer, under the References folder—we are now ready to go.

Getting Started

When working with the Crystal Reports Engine namespace, there is one class you will use most. The `ReportDocument` class is used to represent the report itself and contains all of the properties, methods, and events that allow you to define, load, export, and print your report.

So before we go any further, we need to start building our sample application form and then use the `ReportDocument` class to load a report to use during the course of our discussion about the Crystal Report Engine.

Building the Sample Application Form

We need to draw the Crystal Report Viewer on the bottom of our form. As we work through the different ways to use the Report Engine, we'll use the viewer on the form to view the resulting report.

Chapter 9

We've also disabled the `DisplayGroupTree` property in the Properties window for now to streamline presentation of the methods being used in this chapter. As you like, you can leave this enabled or disabled.

Loading a Report

Before we load the report, we need to have a report added it to our solution, which you did earlier when you added the `employee_listing.rpt` file.

To add the report to your form, switch to the Design view of the default form that was created with your project (`Form1`). In the toolbox, on the left-hand side of the form designer, there is a tab marked Components, and, within that tab, there should be a `ReportDocument` component that you can drag onto your form.

When you drag the `ReportDocument` component onto your form, it should open a second dialog and allow you to select a report document from a drop-down list. The employee listing report should appear here. Once you have selected a report, another section will appear underneath your form that shows the components you are using. `ReportDocument1` should be here.

With that out of the way, we are ready to declare and load our report for use with the Report Engine. Double-click your form to show the code view, and locate the code for the `Form_Load` event—it is here that we are going to declare our report:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
    Dim myReport As New employee_listing()
    CrystalReportViewer1.ReportSource = myReport
    myReport.Load()
End Sub
```

That is all there is to it—once you have loaded a report, you have access to all of the properties, methods, and events associated with the `ReportDocument` object model. For example, if you wanted to extract the title of your report from the `SummaryInfo`, (in the Report Designer, right-click, and select `Report → Summary Info` to set this information) you could simply peek into the `ReportDocument` object model to grab the `SummaryInfo` collection (of which the `ReportTitle` is a member) by inserting this line of code at the end of the `Form_Load` event previously listed:

```
MsgBox(myReport.SummaryInfo.ReportTitle.ToString())
```

There are actually five members of the `SummaryInfo` class:

Property	Description
<code>KeywordsInReport</code>	For returning or setting keywords
<code>ReportAuthor</code>	For returning or setting the report author
<code>ReportComments</code>	For returning or setting the report comments
<code>ReportSubject</code>	For returning or setting the report subject
<code>ReportTitle</code>	For returning or setting the report title

*All of these properties can be viewed or set at run time, but remember, to actually write to the report file with these settings in place, you would need to use the **SaveAs** method, discussed a little later in this chapter.*

Using the Initialization Event

Now, in our previous example, when we loaded the report, we had no way of knowing whether or not the report had been loaded. Luckily for us, there is an initialization event that is fired whenever a report is loaded.

We can use this initialization event to actually tell us when the report was loaded:

```
Private Sub report_InitReport(ByVal sender As Object, ByVal myEvent As  
    System.EventArgs) Handles employee_listing1.InitReport  
    MsgBox("Report loaded fine")  
End Sub
```

This will save us time later when we try to troubleshoot our applications. If we know the report has been loaded safely, that is one less thing to check when problems occur.

Printing and Exporting

Within the Crystal Reports Engine, there are a number of different ways you can produce report output, even without the Crystal Report Viewer. This functionality provides an easy way for you to print directly from your application or print batches of reports without any user intervention.

We are going to start looking at this type of functionality with a simple print application by building on the sample application we are working with.

Printing Your Report

To print your report from your application, the `ReportDocument` class provides a simple `PrintToPrinter` method that can be used to print the report to the default printer. This method requires four parameters:

- Number of copies
- Collation flag (Boolean)
- Start Page
- End Page

So, adding to the form we were working with earlier, we could create a command button that would print your report to your default printer, as shown in Figure 9-2:

Call the button `Print_Button`, and set the text as Print Report (Direct). Now, double-click the button to open the code behind it.

Chapter 9

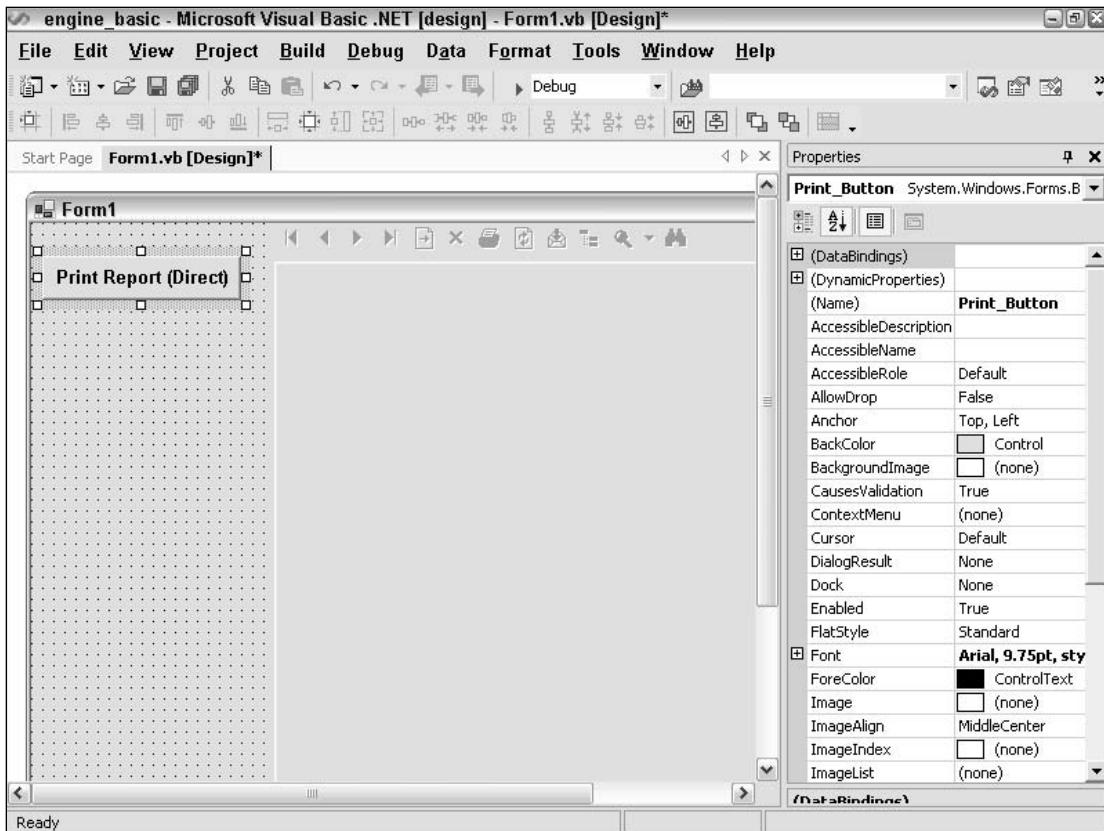


Figure 9-2

In this case, we are going to print one copy of pages 1 to 999 from our report to your default printer. If you double-click the command button you have added to the form, you can add the method call to the code view, and it would look like this:

```
Private Sub Print_Button_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Print_Button.Click
    Dim myReport As New employee_listing()
    myReport.PrintToPrinter(1, True, 1, 999)
End Sub
```

In this case, we used 999, assuming that would be a good upper limit for the number of pages in your report.

If you need a little more control over the print process, there is also a `PrintOptions` class that comes in handy, allowing you to set the printer name to another printer (besides the default) and the number of copies, among other things. This class includes the following properties:

Property	Description
PageContentHeight	Returns the height of the pages content in twips.
PageContentWidth	Returns the width of the pages content in twips.
PageMargins	Returns or sets the page margins collection (including topMargin, bottomMargin, leftMargin, rightMargin).
PaperOrientation	Returns or sets the current printer paper orientation. Options are DefaultPaperOrientation (from the printer), Landscape, and Portrait.
PaperSize	Returns or sets the current paper size. Supports 42 different sizes, including PaperA4, PaperLegal, PaperLetter, and so on.
PaperSource	Returns or sets the current paper source. Supports 13 different paper trays, including Auto (for the printer's automatic selection), Manual, Lower, Middle, Upper, and so on.
PrinterDuplex	Returns or sets the current printer duplex option. Supports options for Default, Horizontal, Simplex, Vertical.
PrinterName	Returns or sets the printer name used by the report.

For a complete list of members in the `PaperSize` class, you can search the Visual Studio .NET Combined Help File for "PaperSize Enumeration." For the `PaperSource` class, search for "PaperSource Enumeration".

A lot of the printer-specific features (`PaperSource`, `PrinterDuplex`) will depend on your printer's capabilities, and you may spend some time trying to figure out which members correspond with the different features on your printer. Usually, a printer will include a technical specification that will include this information, but, if you can't find it (or work it out), check the manufacturer's Web site.

So, to put some of the options of the `PrintOptions` class together and print a report duplex to a specific printer on A4 paper with a new set of margins, the code behind our command button would look something like this:

```

myReport.PrintOptions.PrinterName = ""
myReport.PrintOptions.PaperSize =
    CrystalDecisions.[Shared].PaperSize.PaperA4
myReport.PrintOptions.PrinterDuplex =
    CrystalDecisions.[Shared].PrinterDuplex.Default

Dim myMargins = myReport.PrintOptions.PageMargins
myMargins.topMargin = 10
myMargins.bottomMargin = 10
myMargins.leftMargin = 10
myMargins.rightMargin = 10

myReport.PrintOptions.ApplyPageMargins(myMargins)
myReport.PrintToPrinter(1, True, 1, 999)

```

Chapter 9

If the `PrinterName` string is empty, the default printer is selected.

If you do change the page margins, you will need to use the `PrintOptions` class's `ApplyPageMargins` method to apply your changes.

If you are migrating your code from Visual Basic 6.0, keep in mind that the Crystal Report Engine in Visual Studio .NET no longer supports the `SelectPrinter` method that was so handy in previous versions (it would pop up the standard Select Printer dialog for you). To use this functionality within Crystal Reports.NET, you will need to open the Select Printer dialog yourself, get the name of the printer, and then set the `PrinterName` property of the `PrintOptions` class.

Exporting Your Report

In addition to printing your report without the viewer, you can also export your report without having to use the export button available on the Crystal Report viewer.

Within the `ReportDocument` methods, there is a method called `Export`, which can be used to export directly from your application. Unlike the `PrintReport` method, which would just print the report to the default printer, there are a number of properties that need to be set before you can actually call the `Export` method.

Here is a rundown of all of the properties and objects that are related to the `ExportOption` class:

Property	Description
<code>DestinationOptions</code>	Returns or sets the <code>DestinationOptions</code> object, including <code>DiskFileDestinationOptions</code> , <code>ExchangeFolderDestinationOptions</code> , and <code>MicrosoftMailDestinationOptions</code>
<code>ExportDestinationType</code>	Returns or sets the export destination type
<code>ExportFormatType</code>	Returns or sets the export format type
<code>FormatOptions</code>	Returns or sets the <code>FormatOptions</code> object, including <code>ExcelFormatOptions</code> , <code>HTMLFormatOptions</code> , and <code>PdfRtfWordFormatOptions</code>

So, in another example, we could add another button to our form to export the report, as shown in Figure 9-3:

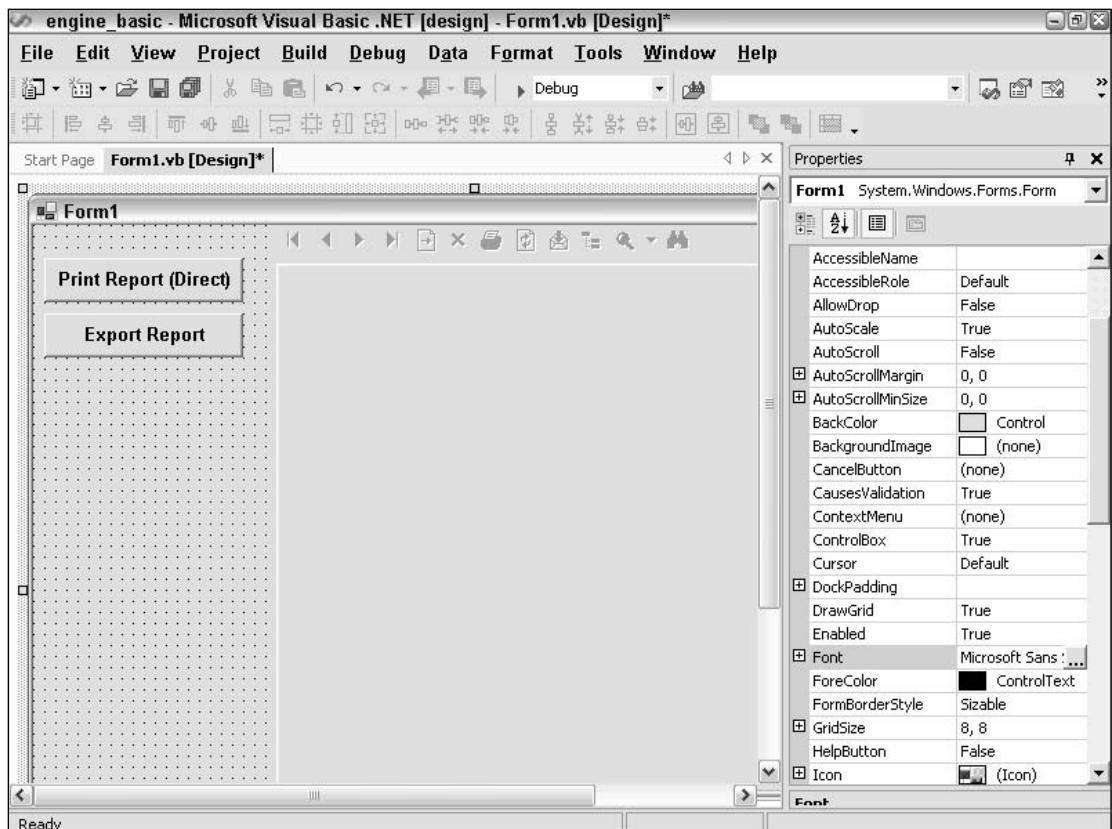


Figure 9-3

Name the button `Export_Button`, and change the `Text` property to `Export Report`.

The code behind the button sets all of the properties and collections of information required. It then uses the `ExportReport` method to export our report. The first thing we need to do in our code is actually set up some variables to hold the different property collections that we will be setting, including properties for the `ExportOptions`, `DiskFileDestinationOptions`, and `FormatTypeOptions`.

```
Private Sub Export_Button_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Export_Button.Click
    Dim myReport As New employee_listing()
    myReport.Load()

    Dim myExportOptions As New CrystalDecisions.Shared.ExportOptions()
    Dim myDiskFileDestinationOptions As New
        CrystalDecisions.Shared.DiskFileDestinationOptions()
    Dim myFormatTypeOptions As New
        CrystalDecisions.Shared.PdfRtfWordFormatOptions()
```

Chapter 9

With some variables created, we now need to select where our exported file is going to be saved and what format it is going to be available in:

```
myDiskFileDestinationOptions.DiskFileName =
    "C:\CrystalReports\Chapter09\test.pdf"
myExportOptions = myReport.ExportOptions

With myExportOptions
    .ExportDestinationType =
        CrystalDecisions.Shared.ExportDestinationType.DiskFile
    .ExportFormatType =
        CrystalDecisions.Shared.ExportFormatType.PortableDocFormat
    .DestinationOptions = myDiskFileDestinationOptions
    .FormatOptions = myFormatTypeOptions
End With
```

Finally, we call the `Export` method to actually export our report:

```
myReport.Export()
MsgBox("Your report has been exported in PDF format and saved to
C:\CrystalReports\Chapter09\test.pdf")
```

When faced with a number of different property collection for destinations, format types and such, it can get a bit messy trying to figure out which combination of properties you need (for example, to export a report to an Exchange folder in RTF format but only the first two pages).

There are actually seven export formats available for Crystal Reports .NET:

- Adobe Acrobat (.pdf)
- Crystal Reports within Visual Studio .NET, Crystal Reports 9.0 (.rpt)
- HTML 3.2 and 4.0 (.html)
- Microsoft Excel (.xls)
- Microsoft Rich Text (.rtf)
- Microsoft Word (.doc)

and three destinations for the exported report:

- Disk file
- Microsoft Exchange public folders
- Microsoft Mail

For more information on the relationship between the objects involved in exporting, as well as the classes and members associated with each of the export formats and destinations, search the Visual Studio .NET Combined Help Collection using the keywords CRYSTAL REPORT EXPORT.

Some of the classes and members that are used with the Crystal Reports Engine are actually part of a `CrystalDecisions.Shared` namespace, which is shared between the Windows Forms Viewer, Web Forms Viewer, and the Crystal Reports Engine to reduce duplication in these namespaces.

Working with Databases

The strength of Crystal Reports .NET is its ability to extract information from a database (or other data source) and present the information in a report that users can view or print so it stands to reason that most of your reports will be based on some database or data source within your application.

The Crystal Reports Engine provides a set of tools for working with databases by giving us the flexibility to change the database login information, location, and other features at run time through the properties and methods associated with the `Database` object.

There are two classes associated with `Database`. They are `Tables` and `Links`. The `Tables` class contains all of the tables that are used in your report, and the `Links` class contains all of the links between these tables as created within the report. Using these two classes, you can set the login information for your database, retrieve or change the location of tables, or change the table linking, among other functions.

We will start looking at these classes with one of the most common developer tasks—specifying the database connection information for your report. If you have established database security, you will want to pass the appropriate username and password for the user who is viewing the report, and the following section will guide you through how this is done.

Logging on to a Database

When creating a report using Crystal Reports .NET, you can include data from multiple data sources in your report. While this feature makes for information-rich reports and can eliminate the need for multiple reports, it does pose a problem when customizing the report at run time.

It would be impossible to set one set of database credentials for all of the data sources in a report so the Report Engine object model caters for these multiple data sources by allowing you to set the connection information for individual tables that appear in your report through the `Tables` class. This class has the following members:

Property	Description
<code>Count</code>	Returns the number of <code>Table</code> objects in the collection
<code>Item</code>	Returns the <code>Table</code> object at the specified index or with the specified name

Chapter 9

Each `Table` object in the `Tables` collection has the following properties:

Property	Description
Fields	Returns the <code>DatabaseFieldDefinitions</code> collection (which we'll look at a little later in this chapter)
Location	Returns or sets the location of the database table
LogOnInfo	Returns the <code>TableLogOnInfo</code> object
Name	Returns the alias name for the database table used in the report

Now, at this point, you are probably wondering how the `TableLogOnInfo` actually gets set. There is a method associated with this class, `ApplyLogOnInfo`, that is used to apply any changes to the database login information for a table.

For collecting and setting the properties relating to `TableLogOnInfo` and connection information, the `CrystalDecisions.Shared` namespace has a `ConnectionInfo` class that has the following properties:

Property	Description
<code>DatabaseName</code>	Returns or sets the name of the database
<code>Password</code>	Returns or sets the password for logging on to the data source
<code>ServerName</code>	Returns or sets the name of the server or ODBC data source where the database is located
<code>UserID</code>	Returns or sets a user name for logging on to the data source

We looked briefly at these properties and methods in Chapter 3, “Designing Reports,” but we didn’t tackle looping through the database. We’ll look at that now.

Drag another button onto your Form, and call it `Database_Button`. Change the Text property to Northwind Report. We’ll create a new Form with this button so right-click the project name, select Add → Add New Item.... and then, out of the dialog that pops up, select Windows Form. The default name will be `Form2.vb`, which is as good as any.

Double-click our new button, and insert the following code:

```
Private Sub Database_Button_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Database_Button.Click
    Dim Form2 As New Form2()
    Form2.Show()
End Sub
```

Now, drag a `CrystalReportViewer` onto `Form2` in the Design mode, right-click the project to Add → Add Existing Item..., Browse to `C:\Crystal.NET2003\Chapter09\worldsales_northwind.rpt` (this location will vary depending on where you have downloaded the sample code to), and add this report to the project.

Working with the Crystal Reports Engine

Next, drag a ReportDocument component onto the Form and, when the dialog opens, select engine_basic.worldsales_northwind.

The next step is to add some additional code to set our ConnectionInfo class.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles MyBase.Load  
    Dim myReport As New worldsales_northwind()  
    CrystalReportViewer1.ReportSource = myReport  
    myReport.Load()  
    Dim myDBConnectionInfo As New CrystalDecisions.Shared.ConnectionInfo()  
  
    With myDBConnectionInfo  
        .ServerName = "localhost"  
        .DatabaseName = "Northwind"  
        .UserID = "sa"  
        .Password = ""  
    End With
```

If you are using a secured Microsoft Access, Paradox, or other PC-type database, the same method can be used, except the **.ServerName** and **.DatabaseName** are left blank.

Then, we can apply this ConnectionInfo by looping through all of the tables that appear in our report:

```
Dim myTableLogOnInfo As New CrystalDecisions.Shared.TableLogOnInfo()  
  
Dim myDatabase = myReport.Database  
Dim myTables = myDatabase.Tables  
Dim myTable As CrystalDecisions.CrystalReports.Engine.Table  
  
For Each myTable In myTables  
    myTableLogOnInfo = myTable.LogOnInfo  
    myTableLogOnInfo.ConnectionInfo = myDBConnectionInfo  
    myTable.ApplyLogOnInfo(myTableLogOnInfo)  
Next  
End Sub
```

In this instance, we are looping through the tables using the table object. You can also loop through the tables through the item and the table name or index.

For instance:

```
myReport.Database.Tables.Item(i).ApplyLogOnInfo()
```

But, it's up to you.

Setting a Table Location

Another handy trick that the Report Engine provides is the ability to set the location for tables that appear in our report. (This is the equivalent of going into the Report Designer, right-clicking, and selecting Database → Set Location.)

Chapter 9

This can be useful for occasions when you have to put historical data into another table or want to separate out data in different tables for different business units, but the structure of the “source” and “target” table have to be the same, or you will get errors when the report is run.

When working with the location of a table, the `Location` property will both return and set where the table resides.

The example that we are now going to build demonstrates how the location of a table in a report could be changed to point to a “current” employee table. In your project, right-click the project name, select Add → Add New Item..., and choose Windows Form. The default name should be `Form3.vb`. Click Open. Drag a button onto the Design view of `Form1.vb`, call the button `Location_Button`, and change the Text property to Set Database Location. Double-click this button, and insert the following code:

```
Private Sub Location_Button_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Location_Button.Click
    Dim Form3 As New Form3()
    Form3.Show()
End Sub
```

We shall use `employee_listing.rpt` to demonstrate the point. This report is already attached to the project so we do not need to add it. However, what we do need to do is to go into our Xtreme database and create a copy of the `Employee` table in Access. This copy should be named `Employee_Current`. Add a few more employee rows onto the end of the table (just so that the information is slightly different), and save it.

There are several versions of Xtreme supplied from various sources, including the ones included with both Crystal Enterprise and Microsoft Visual Studio .NET. Make sure that the version you alter and the data source the report is referencing are the same!

The next thing to do is prepare `Form3.vb`. In the Design view of this Form, drag on a `CrystalReportViewer` and a `ReportDocument` component. When the dialog for the `ReportDocument` comes up, select `engine_basic.employee_listing`.

All that remains is to insert the following code:

```
Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
    MsgBox("Note: To make this sample work, open the Xtreme sample database in
        Access and copy the Employee table to Employee_Current and change some
        values. You should see these changes when you view the report,
        indicating the set location worked correctly")

    Dim myReport As New employee_listing()
    CrystalReportViewer1.ReportSource = myReport
    myReport.Load()

    Dim myDatabase = myReport.Database
```

```
Dim myConnectionInfo As New CrystalDecisions.Shared.ConnectionInfo()
Dim myTableLogonInfo As New CrystalDecisions.Shared.TableLogOnInfo()

Dim myTables = myDatabase.Tables
Dim myTable As CrystalDecisions.CrystalReports.Engine.Table

For Each myTable In myTables

    MsgBox("Before: " & myTable.Location.ToString())

    If myTable.Location.ToString() = "Employee" Then
        myTable.Location = "Employee_Current"
    End If
    myTable.ApplyLogOnInfo(myTableLogonInfo)

    MsgBox("After: " & myTable.Location.ToString())

Next

CrystalReportViewer1.ReportSource = myReport
CrystalReportViewer1.Refresh()
End Sub
```

We're good to go. Run the application, and click the Set Database Location button. Various dialogs should appear that advise you on the changes in the location since the tables cycle through the `For...` loop. Eventually, the report will load, showing the changes you have made.

You could also use this feature to point to a table that resides on a completely different database platform (from SQL Server to Oracle, for example), as long as the table definitions are compatible.

*If you want to ensure that your report has the most recent instance of the data you are reporting from, prior to your export, you can use the **Refresh** method to refresh your report against the database.*

Setting the Join Type

For reports that are based on more than one table, Crystal Reports .NET has a visual linking tool that allows you to specify the links or joins between these tables, as shown in Figure 9-4:

To see this dialog, open the Report Designer, right-click your report, and select Database → Visual Linking Expert....

When working with these tables and links at run time, it can be confusing when working with all of the different elements involved so we'll break it down.

Similarly with `Tables`, there is a `TableLink` object that is contained in a `TableLinks` collection, which has one `TableLink` object for every link that appears in your report.

Keep in mind that tables can have multiple links between them. For example, you may have only two tables, but there may be three key fields that are linked together between those two tables.

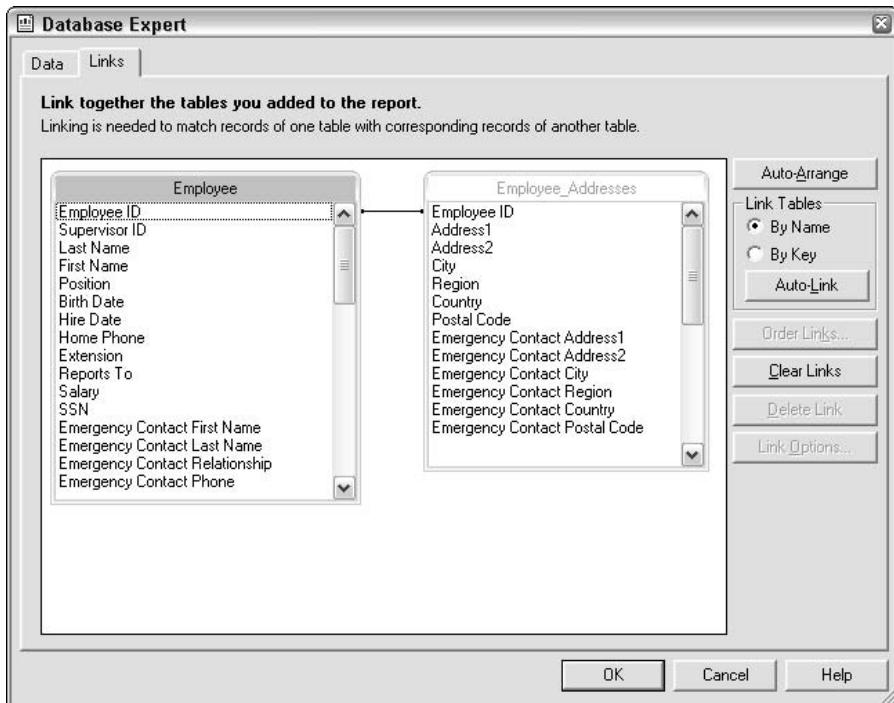


Figure 9-4

A TableLink has the following properties:

Property	Description
DestinationFields	Returns a reference to table link destination DatabaseFieldDefinitions collection
DestinationTable	Returns a reference to the table link destination Table object
JoinType	Returns a summary of the linking used by the table
SourceFields	Returns a reference to table link source
SourceTable	Returns a reference to the table link source Table object

So, to determine the tables and database fields used in linking our tables together, we can loop through all of the links used in our report. We'll look at how we do this now.

Drag another button onto Form1 in the Design view, and name it `Links_Button`. Change the Text property to Show Links. Double-click the button, and add the following code:

```
Private Sub Links_Button_Click(ByVal sender As System.Object, ByVal e As  
    System.EventArgs) Handles Links_Button.Click  
    Dim myReport As New employee_listing()  
    myReport.Load()  
  
    Dim myDatabase = myReport.Database  
    Dim myTables = myDatabase.Tables  
    Dim myTable As CrystalDecisions.CrystalReports.Engine.Table  
  
    Dim myLinks = myDatabase.Links  
    Dim myLink As CrystalDecisions.CrystalReports.Engine.TableLink  
  
    For Each myLink In myLinks  
  
        MsgBox("Destination Table: " & myLink.DestinationTable.Name.ToString &  
            vbCrLf & " ." & myLink.DestinationFields.Item(1).Name.ToString())  
        MsgBox("Source Table: " & myLink.SourceTable.Name.ToString & " ." &  
            myLink.SourceFields.Item(1).Name.ToString)  
        MsgBox("Join Type: " & myLink.JoinType.ToString)  
  
    Next  
End Sub
```

Compile and run. The dialogs should now appear, one after the other, bearing the name of the source and target links and also the join type, as shown in Figure 9-5.

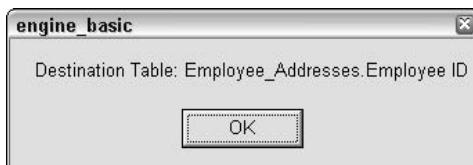


Figure 9-5

Keep in mind that these properties are read-only. You will not be able to set the table linking using these properties. If you do want to change the database linking that is used, you may want to consider pushing the data into the report using a dataset.

Pushing Data into a Report

Earlier in our discussion of the different ways you could deploy a report in Chapter 1, "Crystal Reports .NET Overview," we looked at "Push" and "Pull" type reports. Up until this point, we have been working exclusively with "Pull" reports in which we pull the information from the database and display it in our report.

For "Push" reports, you actually create the report the same way, except that, when the report is run, you can "Push" a dataset to the report, as we did in Chapter 6, "Creating SML Report Web Services." This works in a similar manner to actually setting the data source for an individual table, but, instead of setting the property equal to another table, we are going to set it equal to another data source. In the

Chapter 9

following example, we are using our sample report that we have been working with, but, instead of data from the Xtreme sample database, we are actually connecting to the Northwind database on SQL Server to get the data we need.

```
Dim query = "select * from Customer"

Dim MyOleConn As New System.Data.OleDb.OleDbConnection(conn)
Dim MyOleAdapter As New System.Data.OleDb.OleDbDataAdapter()
Dim MyDataSet As Data.DataSet

MyOleAdapter.SelectCommand = New System.Data.OleDb.OleDbCommand(query, MyOleConn)
MyOleAdapter.Fill(MyDataSet, "Customer")

myReport.Database.Tables.Item("Customer").SetDataSource(MyDataSet)
```

So, instead of actually changing the links and tables that are used within the report, we are actually just pushing another set of data into those structures. As you work with this feature, you are going to pick up some tricks along the way, and one of the handiest tricks is to use a SQL command as the basis of your report. This makes pushing data into the report easier since a report based on SQL command treats the resulting data as if it were one big table.

From your application, you can then get a dataset that matches the fields in your SQL command and then push the data into that one table (instead of having to loop through multiple tables).

Working with Report Options

Another basic task when working with data sources and Crystal Reports is the setting of some generic database options for your report, which are set using the `ReportOptions` class that relates to the report you are working with. Some of these options correspond to the options available in the report designer when you select Designer → Default Settings, but a few (like `EnableSaveDataWithReport`) are not available in the Report Designer, only through the object model.

Property	Description
<code>EnableSaveDataWithReport</code>	Returns or sets the Boolean option to automatically save database data with a report
<code>EnableSavePreviewPicture</code>	Returns or sets the Boolean option to save a thumbnail picture of a report
<code>EnableSaveSummariesWithReport</code>	Returns or sets the Boolean option to save the data summaries you create with the report
<code>EnableUseDummyData</code>	Returns or sets the Boolean option to use dummy data when viewing the report at design time (Dummy data is used when there is no data saved with the report)

A common use of these types of properties is for saving a report with data to send to other users. These properties can be used in conjunction with the `Refresh` and `SaveAs` methods to save a report with data that can be distributed to other users.

To test this, just alter the code in Print_Button code in Form1 as follows:

```
myReport.Export()
myReport.ReportOptions.EnableSaveDataWithReport = True
myReport.Refresh()
myReport.SaveAs("c:\CrystalReports\Chapter09\saved.rpt",
    CrystalDecisions.Shared.ReportFormat.VSNetFileFormat)

MsgBox("Your report has been exported in PDF format and saved to
C:\CrystalReports\Chapter09\test.pdf and your original report has been
saved to C:\CrystalReports\Chapter09\saved.rpt")

End Sub
```

Even if the user doesn't have Crystal Reports or Crystal Reports .NET, a simple viewer application created with Visual Studio .NET is all you need to view the report in its native format (or, if you export to PDF, the Acrobat viewer). The code for a sample viewer is included in the code samples for this chapter.

Setting Report Record Selection

When working with Crystal Reports, you will probably want to use record selection to filter the records that are returned. This record selection formula translates to the WHERE clause in the SQL statement that is generated by Crystal Reports.

You can see the record selection formula for a report by right-clicking your report and selecting Report → Edit Selection Formula → Records. This will open the formula editor, as shown in Figure 9-6, and allow you to edit your record selection formula.

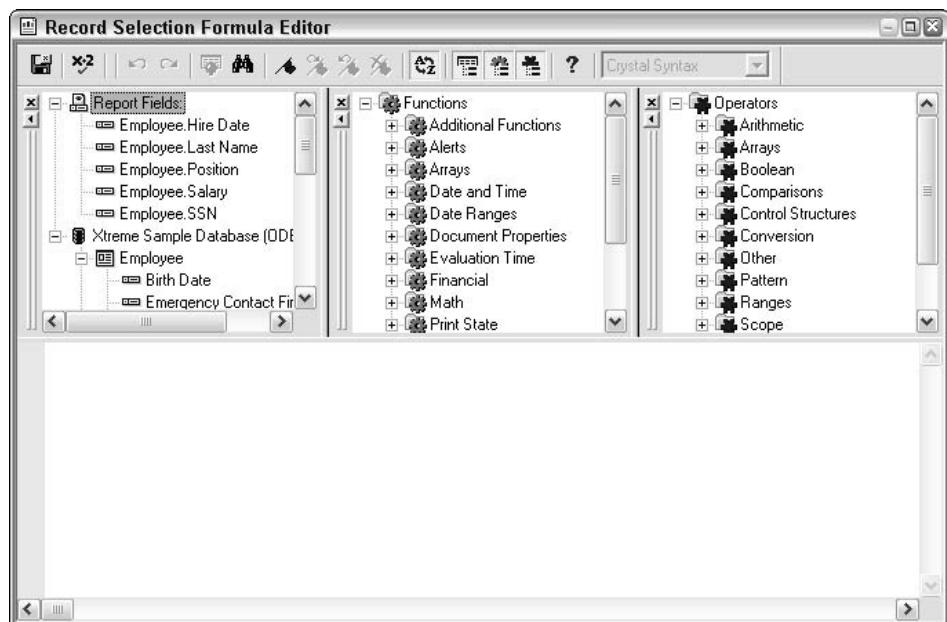


Figure 9-6

Chapter 9

The record selection formula within Crystal Reports is written using Crystal Syntax so you may want to review the section on Crystal Syntax in the previous chapter.

You can retrieve the report's record selection and set it using the same property, as we saw in the examples in Chapter 4, "Report Integration for Windows-Based Applications":

```
myReport.RecordSelectionFormula = "{Employee_Addresses.Country} = 'USA'"
```

Whenever the report is run, this record selection formula will be applied, and the report will be filtered using the formula specified.

*You may look through the object model trying to find where to set the SQL statement that Crystal generates. At this point, your only two options for working with the SQL are to set the record selection (using the method just discussed), which will set the **WHERE** clause, or creating your own dataset using your own SQL statement and then "pushing" the data into the report.*

Working with Areas and Sections

Another often-used class is the `ReportDefinition` class, which is used to retrieve all of the areas, sections, and report objects shown in your report. An area within the context of a Crystal Report corresponds to the types of sections we talked about earlier in Chapter 2, "Getting Started with Crystal Reports .NET." There are seven different types of areas, including:

- Detail
- GroupFooter
- GroupHeader
- PageFooter
- PageHeader
- ReportFooter
- ReportHeader

You may remember from Chapter 2 that all of these different types of areas can also have multiple sections within them (Details A, Details B, and so on), as shown in Figure 9-7:

All of the areas within your report are held within the `Areas` collection of the `ReportDefinition`, which can be accessed through the name of the area or by the number. For example, if you wanted to work with the report header area, you could access it using:

```
myReport.ReportDefinition.Areas("ReportHeader")
```

Likewise, you could access it using its item number as well:

```
myReport.ReportDefinition.Areas(1)
```

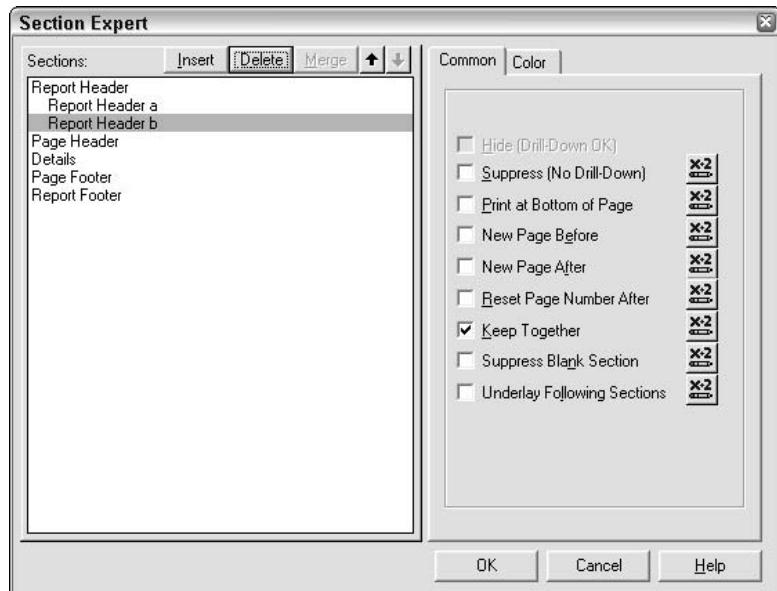


Figure 9-7

So, to start looking at how we can control these areas and sections at run time, we will take a look at areas first.

Formatting Areas

To get a feel for some of the formatting options that are available for sections, as shown in Figure 9-8, open the `employee_listing` report in the Report Designer, right-click a section heading, and select Format Section...:

Most of the formatting options shown in this dialog can be directly read or set using the properties associated with the `AreaFormat` class, including:

Property	Description
<code>EnableHideForDrillDown</code>	Returns or sets hide for drill down option
<code>EnableKeepTogether</code>	Returns or sets the keep area together option
<code>EnableNewPageAfter</code>	Returns or sets the new page after option
<code>EnableNewPageBefore</code>	Returns or sets the new page before option
<code>EnablePrintAtBottomOfPage</code>	Returns or sets the print at bottom of page option
<code>EnableResetPageNumberAfter</code>	Returns or sets the reset page number after option
<code>EnableSuppress</code>	Returns or sets the area visibility

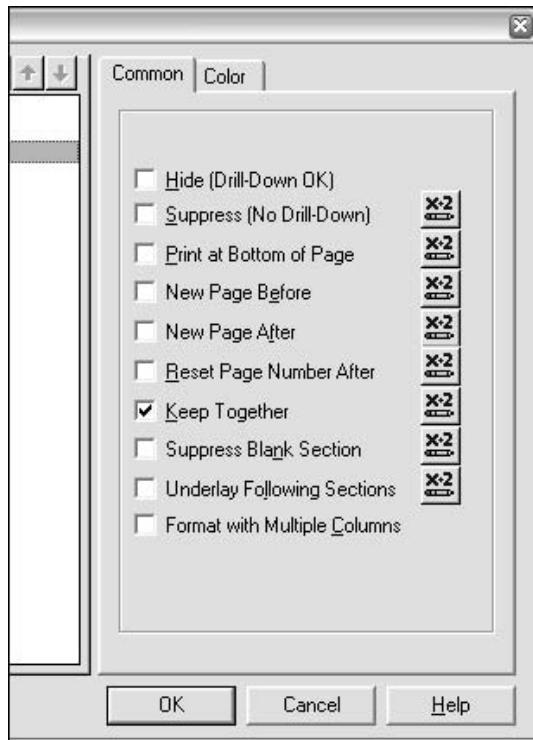


Figure 9-8

So, to format our report so that we are suppressing the page header, the code would look something like this:

```
myReport.ReportDefinition.Areas.Item(1).AreaFormat.EnableSuppress = True
```

Keep in mind that any properties we set for an area also apply for all of the sections within that area (for example, any formatting applied to the "Report Header" would also apply to "Report Header A," "Report Header B," and so on).

Formatting Sections

For sections within an area, we also have a number of properties within a `SectionFormat` class that can control an individual section's appearance, including:

Property	Description
<code>BackGroundColor</code>	Returns or sets the background color of the object using <code>System.Drawing.Color</code>
<code>EnableKeepTogether</code>	Returns or sets the option that indicates whether to keep the entire section on the same page if it is split into two pages

Property	Description
EnableNewPageAfter	Returns or sets the new page after options
EnableNewPageBefore	Returns or sets the new page before option
EnablePrintAtBottomOfPage	Returns or sets the print at bottom of page option
EnableResetPageNumberAfter	Returns or sets the reset page number after option
EnableSuppress	Returns or sets the area visibility
EnableSuppressIfBlank	Returns or sets the option that indicates whether to suppress the current section if it is blank
EnableUnderlaySection	Returns or sets the underlay following section option

All of these properties work just like their counterparts within the `Area` class. The only one that is not Boolean is the `BackGroundColor` property, which is set using the `System.Drawing.Color` palette. If you haven't used this palette before, you may want to review the constants for the different colors available by searching the combined help on "System.Drawing.Color" and looking through its members.

So, to illustrate the use of these properties, we could change the color of our page header and also suppress the page footer for our report. To accomplish this, pull another button onto your form (the final one!) and name it `Format_Button`. Change the `Text` property to `Format Header Color`. In the code behind this button, insert the following:

```

Private Sub Format_Button_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Format_Button.Click
    Dim myReport As New worldsales_northwind()
    myReport.Load()

    myReport.ReportDefinition.Sections.Item(1).SectionFormat.BackgroundColor =
        System.Drawing.Color.AliceBlue
    CrystalReportViewer1.ReportSource = myReport

End Sub

```

Compile and run. The result should be that the section changes color. If you are unsure of what number a particular section is, open the Report Designer and the section numbers will appear on the horizontal divider that marks the top of the section.

Working with Report Objects

Within the sections of your report, there are a number of report objects. You are probably already familiar with these objects since they are the fields, graphs, and cross-tab objects (among others) that appear on your report.

These are:

- `BlobFieldObject`
- `BoxObject`
- `ChartObject`

Chapter 9

- CrossTabObject
- FieldObject
- LineObject
- MapObject
- OlapGridObject
- PictureObject
- SubreportObject
- TextObject

Each of the particular object types within your report has its own unique formatting properties and may also share common formatting options with other types of objects as well.

To determine what type of object you are working with (and subsequently understand what options are available for each type), you can use the `Kind` property of the `ReportObject` to determine the `ObjectType`:

```
If section.ReportObjects(1).Kind = ReportObjectKind.FieldObject Then  
    MsgBox("The first object is a Field Object")  
End If
```

To get started with looking at `ReportObjects`, we are going to look at the most common type, `FieldObjects`.

Formatting Common Field Objects

The main content on a Crystal Report is usually a number of fields that have been inserted and shown on your report. These could be database fields, formula fields, or parameter fields and are used to display the data returned by the report.

When working with these fields at run time, there are two different areas in which we can control the field: the content of the field and the format of the field. As most fields share some of the same formatting options, we will look at the formatting first.

To start with, fields are contained within the `ReportDefinition` object and can be referenced by either the field name or an item number:

```
myReport.ReportDefinition.ReportObjects.Item("Field1")
```

or:

```
myReport.ReportDefinition.ReportObjects.Item(1)
```

You may be tempted to refer to these fields by their name within Crystal Reports (`ReportTitle`), but keep in mind that you can add a field to your report multiple times so, in Crystal Reports .NET, whenever you add a field to your report, a unique number and name are assigned to that field.

You can see the name of the field by looking at its properties within the Crystal Report Designer, as shown in Figure 9-9:

Working with the Crystal Reports Engine

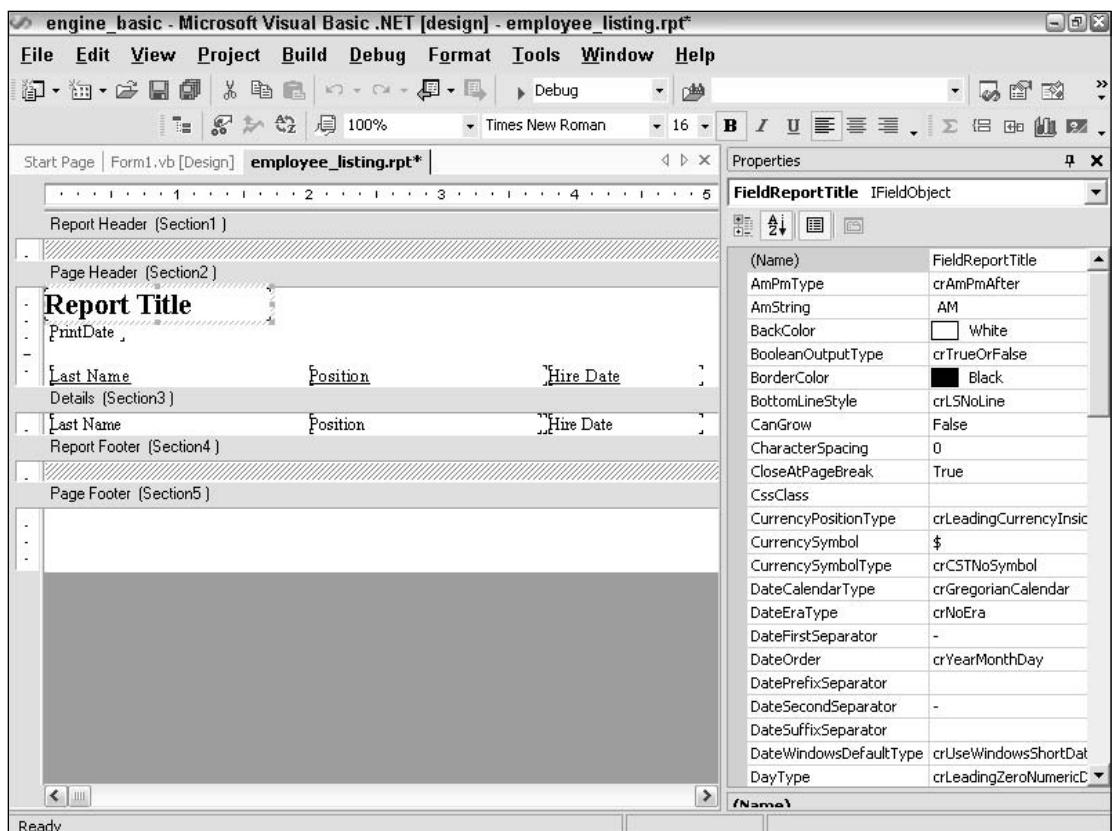


Figure 9-9

If your development follows a set naming convention, you also can change the name of the field to something other than the default Field1, Field2, and so on. Keep in mind that there is no way to change the default naming convention so you may find changing all of the names to be a bit tedious.

To reference the formatting options for a `FieldObject`, we need to access common field properties by referencing the `FieldObject` class, which has the following basic members:

Property	Description
Border	Returns the <code>Border</code> object
Color	Returns or sets the color of the object
DataSource	Returns the <code>FieldDefinition</code> object, which can be used to return and set format information specific to the kind of field
FieldFormat	Returns the <code>FieldFormat</code> object, which can be used to get and set format information specific to the type of field

Table continued on following page

Chapter 9

Property	Description
Font	Returns the Font object (Use the ApplyFont method to apply the changes)
Height	Returns or sets the object height
Left	Returns or sets the object upper-left position
Name (inherited from ReportObject)	Returns the object name
ObjectFormat	Returns the ObjectFormat object that contains all of the formatting attributes of the object
Top	Returns or sets the object upper top position
Width	Returns or sets the object width

So, we can set some of the common formatting properties (such as font and color) directly, as shown in the code example here. It's not included in the sample application, but you should play around with these properties to see what they can do:

```
If section.ReportObjects("field1").Kind = ReportObjectKind.FieldObject Then  
    fieldObject = section.ReportObjects("field1")  
    fieldObject.Color = Color.Blue  
End If
```

There are specific properties that apply to the FieldFormat, depending on what type of field you are working with. When you retrieve the FieldFormat, you will be able to set options that are specific to that field. There are five format types (in addition to a "common" type):

Property	Description
BooleanFormat BooleanFieldFormat	Gets the BooleanFieldFormat object
DateFormat DateFormat	Gets the DateFormat object
DateTimeFormat DateTimeFieldFormat	Gets the DateTimeFieldFormat object
NumericFormat NumericFieldFormat	Gets the NumericFieldFormat object
TimeFormat TimeFieldFormat	Gets the TimeFieldFormat object

In the following sections, we are going to look at how to format the different types of fields using their FieldFormat.

Formatting Boolean Fields

With Boolean fields and the BooleanFieldFormat formatting class, there is only one property, OutputType, which can be set to the following values:

Value	Description
OneOrZero	Boolean value to be displayed as a 1 or 0 (1 = True, 0 = False)
TOrF	Boolean value to be displayed as a T or F (T = True, F = False)
TrueOrFalse	Boolean value to be displayed as True or False
YesOrNo	Boolean value to be displayed as Yes or No
YOrN	Boolean value to be displayed as a Y or N (Y = True, N = False)

So, to change a Boolean field that appears on your report from displaying a binary representation to the text True/False, we could set this property to:

```
fieldObject.FieldFormat.BooleanFormat.OutputType =
BooleanFormat.OutputType.TrueOrFalse
```

If you do need to see other values (such as On or Off, or Active or Inactive), you will probably want to create a formula in your report that translates these values for you:

```
If {Customer.Active} = True then "Active" else "Inactive"
```

Formatting Date Fields

Date fields within Crystal Reports have their own unique set of formatting properties that can be viewed in the Report Designer by right-clicking a date field and selecting Format. You can format date fields by example (picking a date format that looks similar to what you want), or you can customize most aspects of the date field using the dialog shown in Figure 9-10:

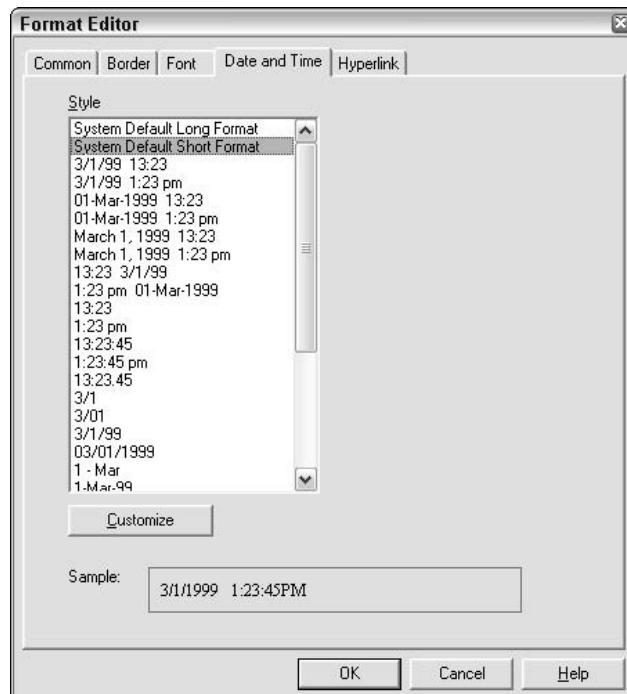


Figure 9-10

Chapter 9

To bring this some of the same functionality at run time, the formatting of a date field has been broken up into multiple classes, which are all members of the `DateFieldFormat` class:

Property	Description
<code>DayFormat</code>	Returns or sets the day format
<code>MonthFormat</code>	Returns or sets the month format
<code>YearFormat</code>	Returns or sets the year format

All of the properties in these classes can be set separately so you don't need to set the month format, for example, if you only want to change how the years are displayed. To start building up a format for a date field within our report, we are going look at the day format first. For the `DayFormat` property, we have three options:

- `LeadingZeroNumericDay`—A single digit day will be printed with a leading zero (for example, 07).
- `NoDay`—A day is not printed.
- `NumericDay`—A day is printed in numeric format with no leading zero.

For the `MonthFormat`, we have five options:

- `LeadingZeroNumericMonth`—The month is printed as a number with a leading 0 for single digit months.
- `LongMonth`—The month is printed as text.
- `NoMonth`—The month is not printed.
- `NumericMonth`—The month is printed as a number with no leading 0.
- `ShortMonth`—The month is printed as text in abbreviated format.

There are also three options for the `YearFormat`:

- `LongYear`—The year is printed in long format with four digits.
- `NoYear`—The year is not displayed.
- `ShortYear`—The year is printed in short format with two digits.

So, to put it all together, here are some examples of how these can produce some commonly requested date formats. To display the date with the days and months with a leading zero and the full four-digit year, the code would look like this. Again, the next two code snippets are for example only and not included in a sample application of their own, but it is recommended that you experiment with these properties:

```
With fieldObject.FieldFormat.DateFormat  
    .DayFormat = DayFormat.LeadingZeroNumericDay
```

```
.MonthFormat = MonthFormat.LeadingZeroNumericMonth  
.YearFormat = YearFormat.LongYear  
  
End With
```

The resulting date field would be displayed as "01/01/2003." For displaying only the month and year, we could change the code to read:

```
With fieldObject.FieldFormat.DateFormat  
  
.DayFormat = DateFormat.NoDay  
.MonthFormat = MonthFormat.LeadingZeroNumericMonth  
.YearFormat = YearFormat.LongYear  
  
End With
```

This in turn would display the date as "01/2003." At this point, you have got to be asking yourself — how do I change the separator character? Or, if you have looked at the formatting properties associated with the field in the Report Designer, you might be wondering how you would set some of the other formatting features.

Unfortunately, the object model does not extend to cover all of the formatting features available for every type of field. If you want to change the format of a particular field and don't see the property listed, you can always create a formula based on the formula field to do the formatting work for you instead.

Formatting Time Fields

For formatting the time fields, the same concept applies, except there is only one class, `TimeFieldFormat`, which has the following properties:

Property	Description
AMPMFormat	Returns or sets the AM/PM type (either <code>AMPMAfter</code> or <code>AMPMBefore</code>) for 12:00am or am12:00
AMString	Returns or sets the AM string
HourFormat	Returns or sets the hour type (<code>NoHour</code> , <code>NumericHour</code> , <code>NumericHourNoLeadingZero</code>)
HourMinuteSeparator	Returns or sets the hour-minute separator
MinuteFormat	Returns or sets the minute type (<code>NoMinute</code> , <code>NumericMinute</code> , <code>NumericMinuteNoLeadingZero</code>)
MinuteSecondSeparator	Returns or sets minute-second separator
PMString	Returns or sets the PM string
SecondFormat	Returns or sets the seconds type (<code>NumericNoSecond</code> , <code>NumericSecond</code> , <code>NumericSecondNoLeadingZero</code>)
TimeBase	Returns or sets the time base (<code>On12Hour</code> , <code>On24Hour</code>)

Chapter 9

So again, by combining all of these formatting properties, you can set the appearance for any time fields that appear in your report. For example, if you wanted to display the time in 24-hour notation, you could simply set the `TimeBase` property, as shown here:

```
fieldObject.FieldFormat.TimeFormat.TimeBase = TimeBase.On24Hour
```

Or, to display any times that would normally be shown as “PM” as “-Evening,” you could set the `PMString` as shown:

```
fieldObject.FieldFormat.TimeFormat.PMString = "-Evening"
```

Which would cause the time field to read “07:13:42-Evening.”

Formatting Date-Time Fields

And finally, for date-time fields, all of the classes available for both date and time fields are consolidated under the `DateTimeFieldFormat` class. The only addition to this class that we haven’t looked at yet is the separator character that will appear between the date and time, which can be set using the `DateTimeSeparator` property, as shown here:

```
fieldObject.FieldFormat.DateTimeFormat.DateTimeSeparator = "="
```

If your report uses date-time fields and you would prefer not to see the date or time component, there is a setting available within the report designer to handle the way date-time fields are processed. Within the Report Designer, right-click the report and select `Report → Report Options` to open the dialog shown in Figure 9-11:

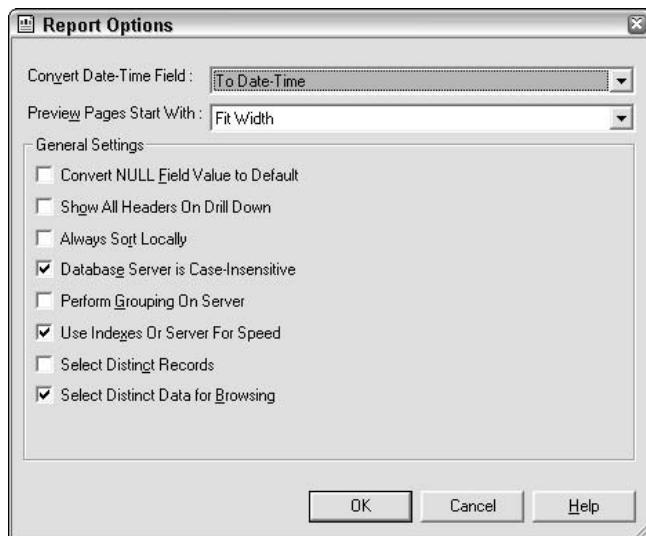


Figure 9-11

Use the first drop-down list to select how date-time fields should be interpreted in your report. You can also set this option globally by right-clicking the report in the Report Designer and selecting `Designer → Default Settings`. Within the Reporting tab, there is an option for converting date-time fields.

Formatting Currency Fields

Currency fields within Crystal Reports have a number of formatting properties that can be set to create financial reports, statements, and other fiscal information and display the data in the correct format for the type of report that is being created.

You can format a number or currency field in your report by right-clicking the field in the Report Designer and selecting Format, which will open a dialog that will allow you to format the field, and show a sample of the field in whichever format is chosen from the list. Or, you can click the Customize button to control the granular properties associated with formatting, as shown in Figure 9-12.

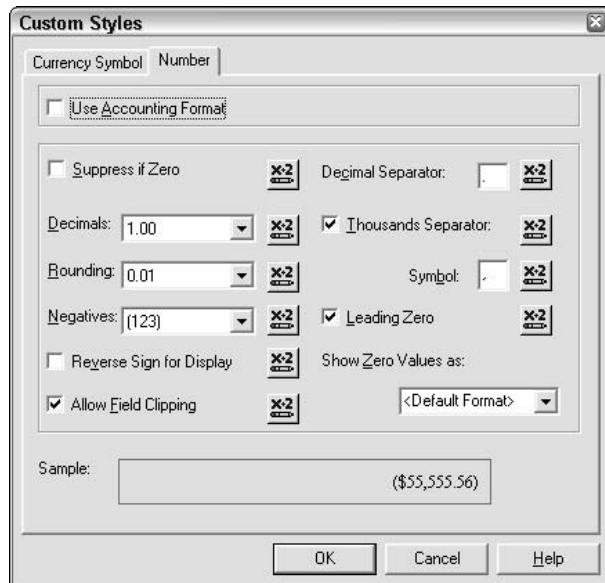


Figure 9-12

Unfortunately, not all of these properties can be changed programmatically, but the following properties of the `NumericFieldFormat` class are supported:

Property	Description
<code>CurrencySymbolFormat</code>	Returns or sets the currency symbol type (<code>FixedSymbol</code> , <code>FloatingSymbol</code> , or <code>NoSymbol</code>).
<code>DecimalPlaces</code>	Returns or sets the number of decimal places
<code>EnableUseLeadingZero</code>	Returns or sets the option to use a leading zero for decimal values
<code>NegativeFormat</code>	Returns or sets the negative format type (<code>Bracketed</code> , <code>LeadingMinus</code> , <code>NotNegative</code> , <code>TrailingMinus</code>).
<code>RoundingFormat</code>	Returns or sets the rounding format type (To see all of the different types available, search the Visual Studio Combined Help for "RoundingFormat Enumeration")

So, to change the format of a numeric field, showing a fixed currency symbol with two decimal places and rounding to the second decimal place (tenth), the code would look like this (again, for illustration only and not included in the sample application):

```
With fieldObject.FieldFormat.NumericFormat  
    .CurrencySymbolFormat = CurrencySymbolFormat.FixedSymbol  
    .DecimalPlaces = 2  
    .RoundingFormat = RoundingFormat.RoundToTenth  
  
End With
```

Keep in mind that we don't have the ability to actually change the currency symbol itself through the object model. You will need to set this in the report design itself or use a formula (for instance, `If {Customer.Country} = "USA" then "US$" else "UKP"`) and position the formula immediately before the numeric field (or use yet another formula to concatenate the currency symbol formula and the field itself together).

Customizing Report Fields at Run Time

Another handy feature that is often overlooked is the ability to customize the report fields at run time. This feature makes it possible to create “generic” reports with blank formula fields laid out on the report design and then specify which database fields should be used to fill these fields at run time.

But, before we can dive in to learning this technique, we will need to create a new project within our solution. To create a new project, select File → New → Project and, in this instance, create a Windows application. We will call the project `engine_customize` because, in the following sections, we are going to be looking at customizing your report fields at run time.

We need a report object to work with. To add the sample reports to your project, select Project → Add Existing Item and select the folder where you unzipped the sample project. You will also need to change the file extension from VB Code Files to All files to see the report files we will be using in this walk-through, which is named `Crystal.NET2003\Chapter09\customer_listing_blank.rpt`.

With a report to work with, we now need to reference the Report Engine so we can use it in our application. This reference may well add itself when you add the report to the project, but, if not or if you are working with one of your custom reports, this will have to be added in Visual Studio .NET. To add a reference, select Project → Add Reference.

Highlight the `CrystalDecisions.CrystalReports.Engine` namespace, and click the Select button and then the OK button to add the reference. The reference to the Crystal Report Engine should now appear in your project in the Solutions Explorer, under the References folder. We are now ready to go.

If you have a look at the report we have added to our solution, it features a very simple design, as shown in Figure 9-13.

To create the report, I have created six blank formula fields and placed them across the page. If you were to preview this report as it stands now, it would be a blank page. In order to specify which fields need to be shown in these formula fields, we need to pass the database table and field name using the Report Engine.

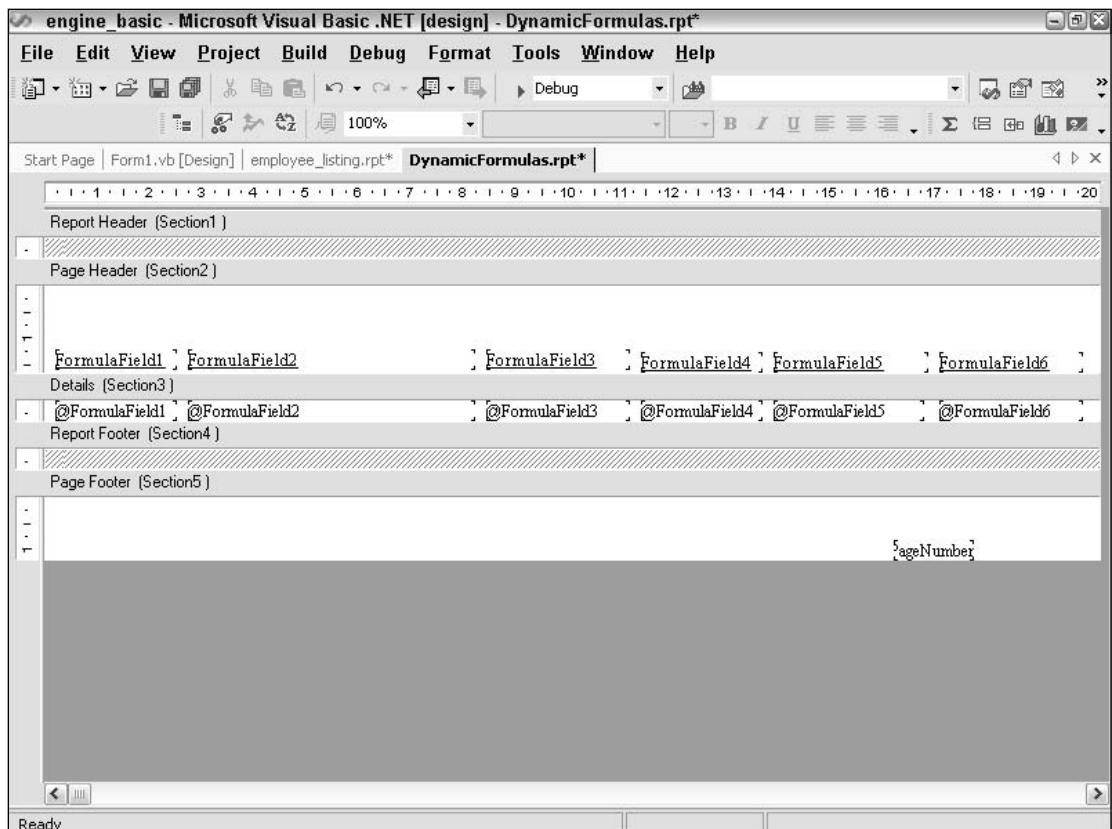


Figure 9-13

As part of the Report Engine object model, there is a `DataDefintion` container that holds all of the formula fields that are present within your report. Within that container is a collection called `FormulaFieldDefinitions` where there will be a singular `FormFieldDefinition` present for each formula in your report. We can pass the database table and field name to the report using the `Text` property of the `FormulaFieldDefinition` (so, in essence, we are writing Crystal Reports formulas on the fly).

The following code illustrates how to access the `FormulaFieldDefinitions` collection and set the formula field text. In this example, we have hard-coded the database table and field names, but you could just as easily prompt the user for what fields they wanted to see in the report and pass this information along.

```
Dim myReport As New CrystalReport1()

Dim myFormulas As FormulaFieldDefinitions

Dim FormulaField1 As FormulaFieldDefinition
Dim FormulaField2 As FormulaFieldDefinition
Dim FormulaField3 As FormulaFieldDefinition
```

Chapter 9

```
Dim FormulaField4 As FormulaFieldDefinition
Dim FormulaField5 As FormulaFieldDefinition
Dim FormulaField6 As FormulaFieldDefinition

myFormulas = myReport.DataDefinition.FormulaFields

FormulaField1 = myFormulas.Item(0)
FormulaField2 = myFormulas.Item(1)
FormulaField3 = myFormulas.Item(2)
FormulaField4 = myFormulas.Item(3)
FormulaField5 = myFormulas.Item(4)
FormulaField6 = myFormulas.Item(5)

FormulaField1.Text = "{Customer.Customer ID}"
FormulaField2.Text = "{Customer.Customer Name}"
FormulaField3.Text = "{Customer.Phone}"
FormulaField4.Text = "{Customer.City}"
FormulaField5.Text = "{Customer.Region}"
FormulaField6.Text = "{Customer.Country}"

CrystalReportViewer1.ReportSource = crReport
```

When your report is printed, it will look something like the report in Figure 9-14:

FormulaField1	FormulaField2	FormulaField3	FormulaField4	FormulaField5	FormulaField6
1	City Cyclists	810-939-6479	Sterling	MI	USA
2	Pathfinders	815-756-9831	DeKalb	IL	USA
3	Bike-A-Holics Anonymous	614-759-9924	Blacklick	OH	USA
4	Psycho-Cycle	205-430-0587	Huntsville	AL	USA
5	Sporting Wheels Inc.	619-457-3186	San Diego	CA	USA
6	Rockshocks for Jocks	512-349-7705	Austin	TX	USA
7	Poser Cycles	612-947-4320	Eden Prairie	MN	USA
8	Spokes 'N Wheels Ltd.	515-237-7769	Des Moines	IA	USA
9	Trail Blazer's Place	608-273-4883	Madison	WI	USA
10	Rowdy Rims Company	805-375-0117	Newbury Park	CA	USA
11	Clean Air Transportation Co.	610-941-5771	Conshohocken	PA	USA
12	Hooked on Helmets	612-947-3344	Eden Prairie	MN	USA
13	C-Gate Cycle Shoppe	540-662-7167	Winchester	VA	USA
14	Alley Cat Cycles	508-287-8535	Concord	MA	USA
15	The Bike Cellar	540-662-2540	Winchester	VA	USA
16	Hercules Mountain Bikes	815-756-1819	DeKalb	IL	USA
17	Whistler Rentals	503-647-3363	Tualatin	OR	USA
18	Bikes and Trikes	810-939-2133	Sterling	MI	USA
19	Changing Gears	714-450-7009	Irvine	CA	USA
20	Wheels and Stuff	813-539-9862	Clearwater	FL	USA
21	Uni-Cycle	614-759-8634	Blacklick	OH	USA
22	Crank Components	217-359-3547	Champaign	IL	USA
23	Corporate Cycle	515-237-9995	Des Moines	IA	USA
24	Pedal Pusher Bikes Inc.	604-941-4617	Port Coquitlam	BC	Canada
25	Extreme Cycling	813-539-4247	Clearwater	FL	USA

Figure 9-14

Don't forget that, when you do pass the table and field name to the Report Engine, we will need to use the standard Crystal Reports format for displaying database fields with the convention of `tablename.fieldname`, enclosed in curly braces (for example, `{Customer.Country}`).

And, you can also use this same technique to control Grouping, Sorting, and more—anywhere a formula field can be used. To use this technique with Grouping, for example, create a formula field called "MyGroup" that references a single database field, and insert a group based on this formula field. You can then use the technique described previously to control which database field is used for grouping. And, if you consider that you could create multiple formulas and groups, you have a flexible solution for user-customized reports.

Summary

The Crystal Report Engine encompasses a lot of the functionality you will need when it comes time to integrate reporting into your application. Throughout the chapter, we have looked at some of the most common uses of the Report Engine and have hopefully built a foundation in which you should be able to feel comfortable applying the same concepts to other areas of the Report Engine's functionality.

In our next chapter, we will be looking at how to manage the distribution of applications that use Crystal Reports and examining some of the issues that may arise out of such a distribution.

10

Distributing Your Application

Finally, with your development and testing finished, one of the last steps in the software development lifecycle is the actual deployment of your application to the end users.

In this chapter, we will look at the tools Visual Studio .NET provides to help distribute applications and how these tools can be used to distribute applications that integrate Crystal Reports. This will include:

- Distribution overview
- Deploying Windows applications
- Deploying Web applications

This chapter has been designed so that, if you are interested in deploying Windows applications, you can turn immediately to that section and get started. Likewise, if you are developing Web applications, there is a separate section for Web deployment. (This section is not comprehensive to avoid repeating details that can be found in the section covering Windows applications.)

By the end of this chapter, you will be able to identify the setup and distribution tools within Visual Studio and understand how they can be used to package and distribute your application. You should also be able to create a setup program from an application that integrates Crystal Reports and successfully install the same application on a target machine.

Distribution Overview

If you come from a background in which you have developed Visual Basic 6.0 applications, you will be familiar with the Packaging and Deployment Wizard that was available with the Enterprise

editions of the product and may even have used these tools to create script-based setups for your own applications.

It probably only took one or two attempts at distributing reporting applications with the Packaging and Deployment Wizard to realize that there had to be a better way of doing things. To start with, it was difficult to determine which run-time files needed to be included with your report, and then, once those same files were identified, you had to ensure that the files didn't conflict with or overwrite any other files on the target system.

Developers often added every Crystal-related file they could find to the setup (just to be on the safe side), but the size of the setup program and distributable files would blow out of proportion to the functionality Crystal Reports provided. Clearly, there had to be a better way of distributing reporting applications.

One of the design goals of Visual Studio .NET was to introduce a new set of powerful, integrated tools for developers that provided the majority of the functionality required to create and distribute applications. With that in mind, Visual Studio .NET enhances the ability to create deployment projects using Windows Installer technology that can generate an `.msi` file for your application; this provides a manageable framework for distributing your application.

For more information on Windows Installer, check out the MSDN library, and search under ".NET Framework Deployment" or "Windows Installer."

To get us started, we are going to take a look at some of the deployment tools within Visual Studio .NET and then, later in the chapter, see how they can be used to deploy our own applications.

Getting Started

To start, the tools available for creating setup program and deployment are now integrated within the Visual Studio IDE. You may have noticed that, within Visual Studio .NET, there are some new project types. If you select File → New → Project and choose the folder for Setup and Deployment Projects, you can see that we have a number of different templates available for use when creating setup projects.

Setup Projects

If you are developing Windows-based applications, you could use a Setup Project to create a Windows Installer file for your application. Using this, you could deploy your application to a number of different Windows platforms and allow users to install all of the files required by your application on their local machine.

Web Setup Projects

For Web-based applications, there is also a Web Setup Project that can be used to create a Windows Installer file but with a different target. For Windows applications, the associated files are usually installed on the local machine. For Web-based applications, the required files will be installed in a virtual directory on (or accessible by) your Web server.

Merge Module Project

Merge modules provide a container for application components and make it easy to distribute a number of files at once. An example of where a merge file could be used is with report distribution. You could create a merge file with all of your reports and simply add the merge file to our setup program (instead of adding them as individual reports).

Merge modules have a number of inherent benefits, including version control and portability between projects. When you are ready to update your application with a new version of all of the reports, you could simply create another merge module and include that in your setup program, making a neat transition between different versions of files.

Crystal Reports .NET includes a number of merge modules for its own framework and run-time files that we will be using a little later with our own setup projects.

The creation and distribution of Merge modules and CAB files is outside of the scope of this book, but you can find more information in the Visual Studio .NET Environment documentation within the combined help collection.

Setup Wizard

To make things a bit easier, there is a Setup Wizard that can guide you through creating a setup project for your application. The Setup Wizard can be selected through the File → New → Project menu and then choosing the Setup and Deployment Projects folder.

The Setup Wizard will guide you through the creation of your project, selecting a project type, and the files that are to be distributed. When you have finished, you will have a new setup project that you can customize as required.

Before we get into actually working with these different types of projects and the walkthroughs for this chapter, we are going to look at some of the basic deployment requirements.

Basic Deployment Requirements

To start, applications created in Visual Studio .NET require that the .NET Framework is installed prior to the application installation. The installer we are going to create actually needs the .NET Framework itself.

Microsoft provides a redistributable file (Dotnetfx.exe) that will need to be installed before you install your setup that includes its own merge module. This module is not used for distributing the framework, just for checking that it is has been installed.

The .NET Framework contains all of the underlying system files that make it possible for .NET applications to run on your computer. For operating systems released after Visual Studio .NET, the .NET Framework will already be installed.

Chapter 10

A launch condition (which we will look at later) called `MsiNetAssemblySupport` automatically gets added to any setup project you may create. Its sole purpose is to check for the .NET Framework. If it doesn't exist, it halts the installation. You must have the framework installed from the redistribute file for your application to work.

The most common scenario for installing this redistributable file is either using a batch file or a splash page (through a Web page or other method) for your application, giving a link to both the redistributable and your own setup program with instructions on how to install the framework and subsequently your own application.

Operating System

Windows applications created with Visual Studio .NET and the .NET Framework can be deployed on most Windows operating systems—with the exception of Windows 95 or earlier.

If you plan to deploy Web applications, Windows 2000 Server, or Advanced Server; Windows Server 2003 Web Server Edition (or Advanced or Enterprise edition) is recommended (and of course, IIS!).

Hardware

The minimum hardware requirements for the .NET Framework and your application depend a lot on the type of application you have created and the response time you expect (you wouldn't want a form that pulled back 10,000 rows of data to be on a client machine with 64MB of RAM).

For the minimum requirements for the .NET Framework, you can consult the Visual Studio .NET Framework Developers Guide online, but the following details the recommended requirements, based on experience deploying Windows applications that integrate reporting:

- Processor: Pentium 100MHz or above
- Memory: 128MB or above
- Hard Disk: 80MB available, plus report file size

For Web-based applications, you would probably want to size the server based on the number of concurrent users. For a small workgroup application, the system requirements listed earlier may be adequate. For organization-wide applications, you will probably want to make a significant investment in both the processor and memory.

For large applications that need to scale beyond a single server, you may want to consider moving your application to Crystal Enterprise.

With a quick look at the types of setup projects we can create and the basic requirements for our target client and server machines out of the way, we can jump in to actually creating our first setup.

Deploying Windows Applications

This section details how to create an installer to distribute a simple reporting application.

To get started, we need a simple reporting application to play around with. For our walkthrough, we are going to use the sample Windows project that can be found within the downloadable files for this chapter. In the download files, there is a single solution file for this chapter (`Chapter10.sln`) and, within the solution file, there is a project called `ch10_app`, which contains a sample application that we will be using to demonstrate distributing a Windows application.

The application consists of a single form that hosts the Windows Crystal Report Viewer and, when run, will display a preview of a Customer Listing Report.

Creating a New Setup Project

As we stated earlier, to create a new Setup Project for an application, select `File → Add Project → New Project`, open the `Setup and Deployment Projects` folder, and select `Setup Project` from the available templates. To keep things simple, we have already created the one for our application and have called the setup project `ch10_Setup`, as shown in Figure 10-1.

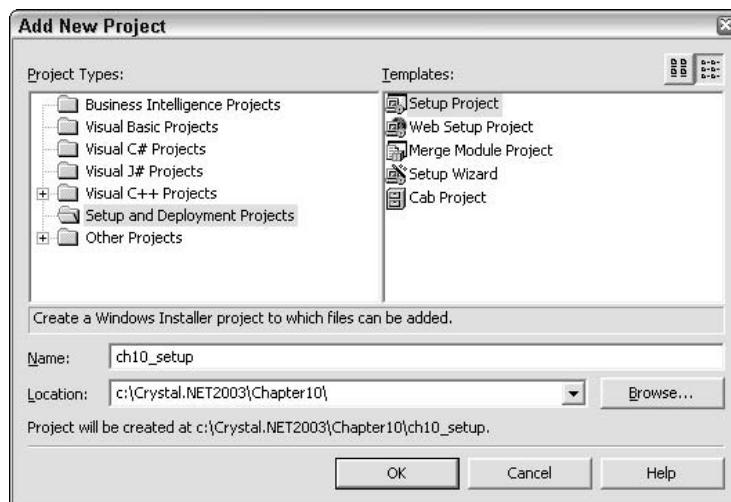


Figure 10-1

This will open the design environment and the File System Explorer, as shown in the Figure 10-2 (which we will come back to).

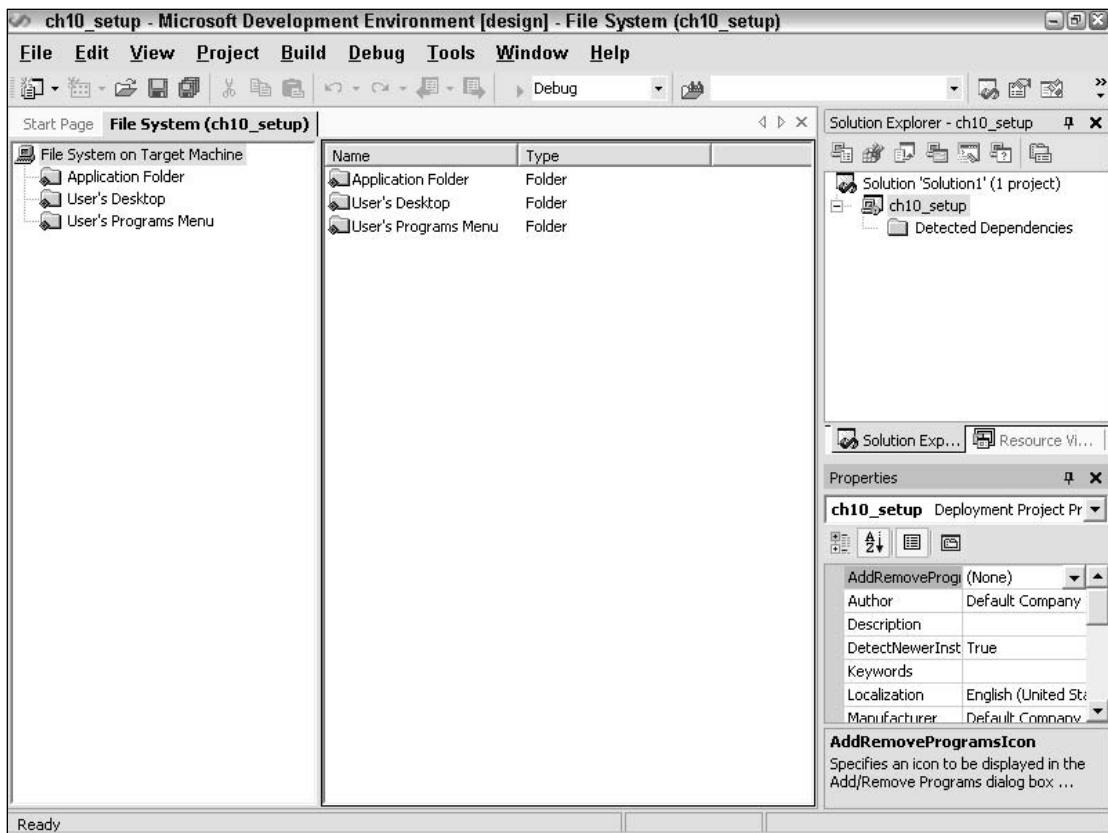


Figure 10-2

For now, we will see how to set some basic properties for our setup project. Select View → Properties Window. In the Solution Explorer, click the name of your setup project, which will open a property page that allows you to enter a number of different properties related to your application.

There are a number of properties available. We won't change any of them here, but the most commonly used ones are:

- Author
- Manufacturer
- Product Name
- Version

The `ProductName` property controls how your application will be advertised on the Add/Remove Programs menu in Control Panel, and other properties are also used for registry entries and other application properties.

Anywhere you see a property in square brackets (for instance, [Manufacturer]), it means that this property is being treated as a variable and its value will be inserted when the application setup is created.

Selecting Project Outputs

When you first create the setup project, the IDE is opened to show the File System Explorer, as shown in Figure 10-3.

The File System Explorer is used to add the various types of files (.exe, .dll, .rpt) to your setup project and, when your setup program is run, the files will be copied to the location you specify.

In the File System Explorer, the majority of files from your project will be targeted at the Application Folder. By default, any items added to this folder will be copied to `drive:\[ProgramFilesFolder\]\[Manufacturer]\[ProductName]`, but you can change this default, if you like, by editing the respective properties in the Properties dialog.

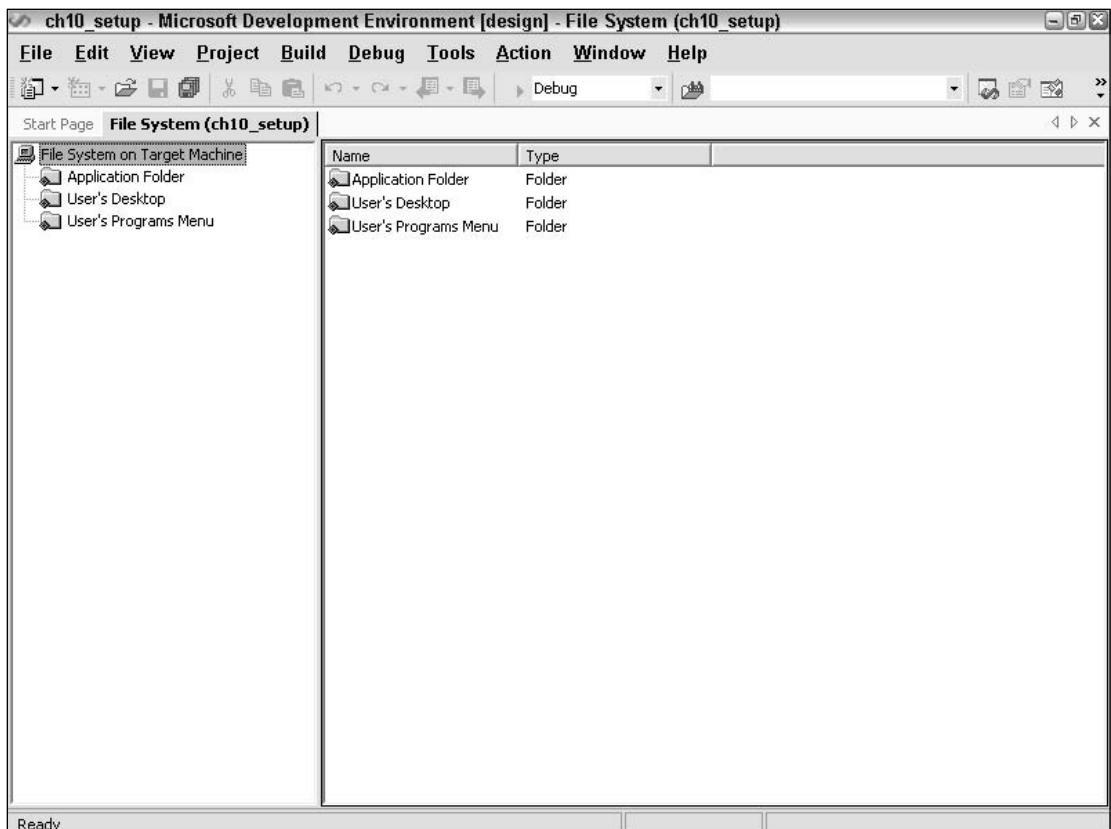


Figure 10-3

To add files to the Application Folder, you would click it and select Action → Add → Project Output; Again, this has already been done for this example. This will open the dialog shown in Figure 10-4 and allow you to add a Project Output Group.

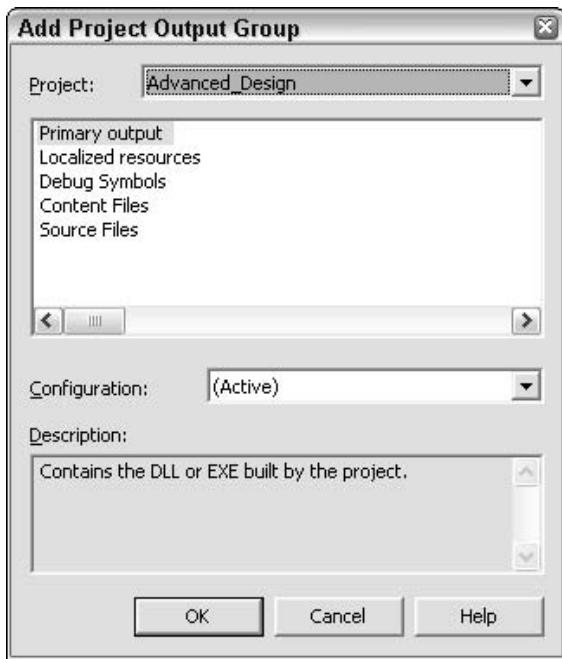


Figure 10-4

If you select Primary output, this will add the primary .exe and .dll files from your application. You would need to repeat the process and select Content Files to actually pick up the report files within your application.

Determining Run-Time File Requirements

With the main components of your application added to the setup project and all of the report files as well, you would now need to determine which run-time files are required to make those same reports work on the target machine.

When looking at the run-time file requirements for Crystal Reports, there are a number of considerations, including the data source for your report (and any drivers in use) and export formats you wish to support, among other things.

There are four different types of merge files that can be used to help you determine these requirements:

- ❑ *Crystal_Managed2003.msm*—The managed component MSM handles the distribution of all the managed components, which include the Windows Form Viewer, Web Forms Viewer, and all of the Crystal Decisions namespaces.

- Crystal_Database_Access2003.msm—The database access MSM handles the distribution of all of the other files that are needed to get the reports to run. This includes the database, export, and charting drivers.
- Crystal_Database_Access2003_enu.msm—The MSM handles language-specific components.

If you are deploying .NET applications using other languages besides English, you will have different versions of this merge module for your language.

- Regwiz.msm—Handles the installation of the Crystal Decisions keycode so that your users are not asked to register their versions of Crystal Reports when viewing reports.

If you use ADO .NET data with reports in your application, there are also some additional merge modules you may need to successfully distribute your application, including VC_User_CRT71_RTL_X86.msm and VC_User_STL71_RTL_X86.msm. These merge files are used to install MSCVR71.dll and MSVCP71.dll, which are required by the ADO .NET data driver (crdb_adoplus.dll).

These merge files will do most of the work for you when deciding which files should be included in your setup program, but you should always double-check the files that the merge module has specified, just to make sure they have them all.

There are different merge modules to be used for Visual Studio .NET 2002 and 2003. If you need to deploy a Visual Studio .NET 2003 application that includes Crystal Reports, check the Crystal Decisions knowledge base at <http://support.crystaldecisions.com> or the previous edition of this book for the details of those files.

Adding Merge Modules

If you look in the Solutions Explorer under your setup project, these modules will appear in one of two places. If your setup project has detected a dependency that has already been detected (from the files you added earlier), the corresponding merge module can be found in the Detected Dependencies folder, as shown in Figure 10-5.

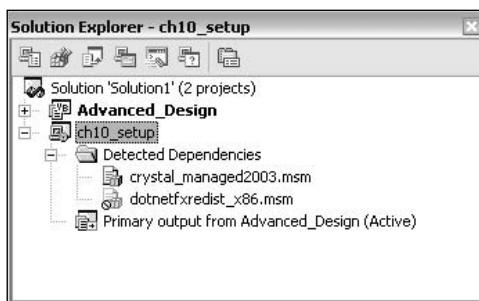


Figure 10-5

If the merge module you require hasn't been automatically detected, you can add a merge module to your setup project by selecting Project → Add → Merge Module and browsing for the merge module you want by using the dialog shown in Figure 10-6.

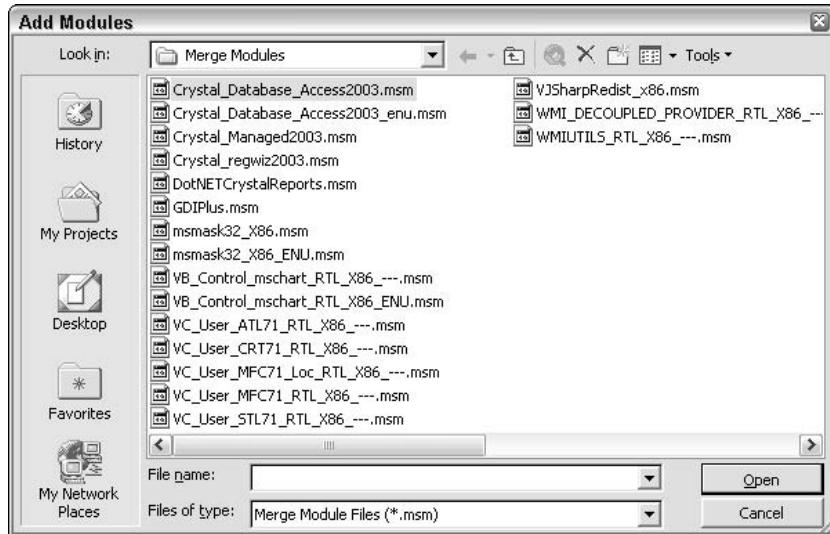


Figure 10-6

Select the merge module you want to add, and click Open to add the module to the list. If you would like to see what files are included with a module, view the Properties and check out the Files property, as shown in Figure 10-7.

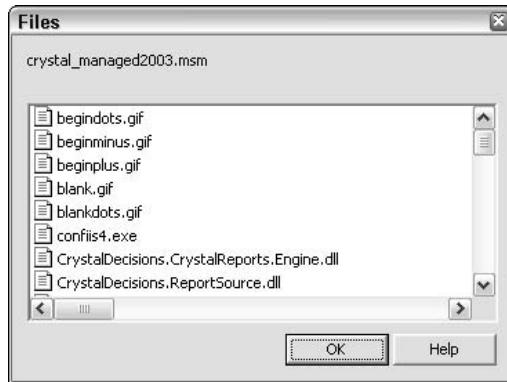


Figure 10-7

You can also see any dependencies that a module has by viewing the `ModuleDependencies` property, which will give the dialog shown in Figure 10-8.

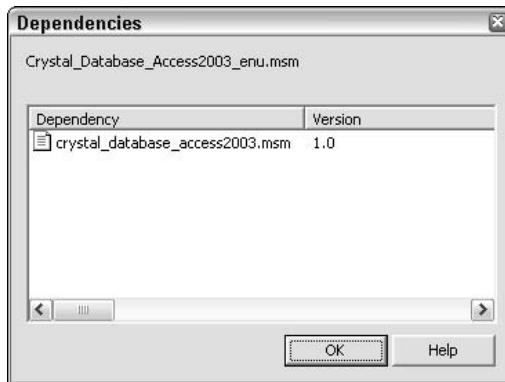


Figure 10-8

Working with Licensing

One of the requirements for using Crystal Reports .NET and distributing the free run time and reports with your application is that you register the software and your personal details with Crystal Decisions. What better way to enforce registration than with a nag screen that appears whenever you open the report designer? (Charming, I know!)

In order to successfully distribute your application and get rid of the nag screen, you are going to need to register with Crystal Decisions and obtain a registration number. When you first started the Crystal Reports designer, chances are you were prompted to register at that point.

If you just clicked Cancel (like most people), you can still register by opening the Report Designer and selecting Register from the right-click menu.

If you have registered, we need to take the registration number you were given and enter it into the Regwiz.msn merge module.

To copy your registration number (or to find out if you are registered), select Help → About Microsoft Development Environment to display a list of all of the Visual Studio .NET products you have installed.

Note the registration number (or to do it the easy way, highlight Crystal Reports .NET and click the Copy Info button). From that point, click OK to return to your project, and locate the License Key property under Regwiz.msm merge module. You will need to enter or paste this license key before you build your setup project.

This is one of the most common errors when deploying applications that use Crystal Reports so don't forget to do it every time you create a setup project.

Building Your Setup Project

The last step of creating our setup project is to actually build the setup project. To build your setup, select Build → Build ch10_Setup and keep an eye on the Output window.

The default Project Configuration is Debug, and Projectname is the name of the deployment project. In our instance, if you had unzipped the sample files for this chapter into a CrystalReports directory on your machine, the setup directory would be found at C:\Crystal.NET2003\Chapter10\code\setup_wizard\Debug.

Along with the MSI file that has been generated, there are also some additional files that should be in the same directory:

- setup.exe—a wrapper for the .MSI file that has been created and a utility that verifies the correct Windows Installer version and installs the correct version
- setup.ini—an .ini file containing the location of the Windows Installer files
- Instmsia.exe—the Windows Installer files for Windows 95, 98, and ME
- Instmsiw.exe—the Windows Installer files for Windows NT

If the setup finds that Windows Installer is not present or the correct version, it will launch the correct executable (instmsia.exe or instmsiw.exe) to install or update the Windows Installer service before installing your own application.

Testing and Deploying Your Setup Project

To test your generated deployment package, copy the entire directory to another computer or CD, and run the setup.exe file.

To test your application setup, you should be able to see where your files were installed and verify that they are present. Also, the application should appear under the Add/Remove Programs option in the Windows Control Panel.

In addition, if you have added a shortcut to your application, you should be able to select the shortcut you have created, and it should launch the application. Make sure that you test the reports themselves. View a number of different reports and try out the features such as drill-down and exporting.

Once you are satisfied the application is installed and that it and the reports run correctly, you can distribute the setup files within the subdirectory to users as required.

Deploying Web Applications

This section details how to create an installer to distribute a simple reporting application. We won't cover the steps in much detail here because most of the information is the same as for Windows applications; so if you have jumped straight to this part of the chapter, then please refer to Deploying Windows Applications to fill in the details.

Preparing Your Web Server

Before you can install a new Web application, the Web server you are installing needs to have the .NET Framework installed first. Just as with Windows applications, there is no automated way to install this from your setup project so you will probably have to create a batch file or install it manually.

In addition to the .NET 1.1 Framework, if your Web application accesses data from a database or other data source, you will need to install MDAC 2.6 or greater in order for your application to work. You can download the latest MDAC components from the Microsoft Web site at www.microsoft.com/data/.

Finally, when exporting directly from Crystal Reports and the Web Forms Viewer, you may need to configure some additional MIME types on your Web server to associate a file extension (such as a PDF file) with its helper application (in this case, Acrobat32.exe).

For more information on configuring MIME types for your version of IIS, visit the MSDN library at <http://msdn.microsoft.com>, and search for "MIME."

Creating the Setup Project

Firstly, just as in the section on Windows deployment, we need a simple Web reporting application to deploy, and one has been included for you in the downloadable files for this chapter in a project named ch10_web_app. This application consists of a single Web Form that has the Web version of the Crystal Report Viewer embedded and allows you to preview the same Employee Listing report that we looked at when working with the sample Windows application earlier.

To see this application working, you will need to create the virtual directory and place the files in this directory.

Again, we need a Setup Project for this Web application. As before, we have already added this to our sample project called ch10_web_setup, as shown in Figure 10-9.

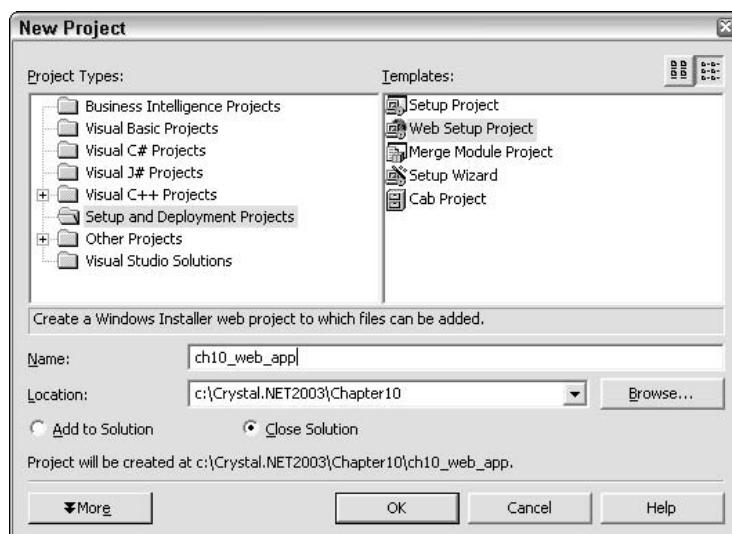


Figure 10-9

Chapter 10

If you need to set some basic properties for your setup project; click the name of your setup project, and select View → Properties Window, which will open the property pages.

There are a number of properties available. The most commonly used ones are the same as for Windows applications, except for Restart WWW Service that controls whether or not the WWW Service for IIS will be restarted when you install your Web application. Whether this option is required or not is up to you and the requirements of the components you are installing on the Web server itself.

After this, you will need to consider selecting the Project Outputs, determining Run-time File Requirements, and adding Merge Modules by following the instructions in the Deploying Windows Applications section. Although these have already been done for this example, you should know about them for your future projects.

Next, we will consider licensing. In order to successfully distribute your applications, you are going to need to first register with Crystal Decisions and obtain a registration number, as we covered in the section on Windows Applications.

Again, just like when deploying Windows Applications, this is one of the most common errors when deploying applications that use Crystal Reports so don't forget to do it every time!

Building Your Setup Project

To build your setup, select Build → Build ch10_web_setup. The default Project Configuration is Debug and Projectname is the name of the deployment project. In our instance, the directory would be ch10_web_setup\Debug\ch10_web_setup.msi.

As we saw when we built our Windows application setup, there are also some additional files that have been generated along with the .msi file, which will be in the same directory. For more details about these files, see Deploying Windows Applications.

Testing and Deploying Your Setup

To test your generated deployment package, copy the entire directory to another computer or CD, and run the Setup.Exe file.

You must have install permissions on the Web server you are using in order to run the installer, and, in addition, you must also have the correct IIS permissions to create a virtual directory and install this application.

To test your Web application setup, you should be able to see where your files were installed on the Web server and verify that they are present. Also, the application should appear under the Add/Remove Programs option in the Windows Control Panel, as shown in Figure 10-10.

To test the application itself, open Internet Explorer and type the URL `http://ComputerName/ch10_web_app`. Make sure that you test the reports themselves. View a number of different reports, and try out the features like drill-down, exporting, and so on.

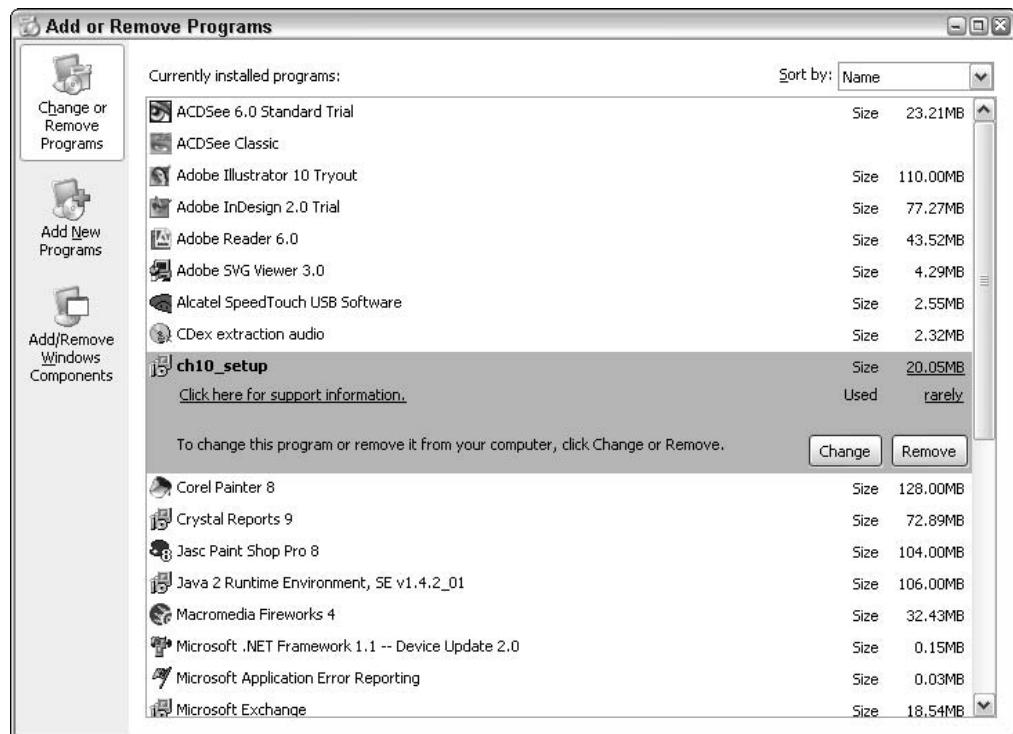


Figure 10-10

Once you are satisfied the application is installed and that it and the reports run correctly, you can distribute the setup files within the subdirectory to users as required.

Summary

In this chapter, we had a look at deploying both Windows and Web applications using the tools available within Visual Studio .NET (and with a little help from Crystal Reports). Throughout the chapter, we looked at some of the tools that are available to make creating setup programs easier and walked through examples of creating setups for both Windows and Web-based applications, including how to build, test, and deploy these setup files. With the skills gained from this chapter, you should be able to successfully deploy and configure reporting applications for one user or for one hundred.

A

Troubleshooting

When working with any development environment or tool, you are going to spend a fair amount of time troubleshooting to diagnose problems, resolve conflicts, and so on. Crystal Reports .NET is no different. This appendix lists some of the resources available to help you troubleshoot your reporting application as well as some common problems you may experience.

Troubleshooting Resources

There are a number of resources available to help you work out any issues or problems you may encounter, and a good place to start is with the resources provided by Crystal Decisions.

Crystal Decisions Knowledge Base

<http://support.crystaldecisions.com/search/>

Crystal Decisions has recently revamped its support Web site with a new comprehensive search engine that allows you to search multiple categories of documents, including the Knowledge Base. The Crystal Decisions Knowledge Base provides a comprehensive selection of articles across the range of Crystal Decisions products and is updated twice a week with over 100 documents relating to Crystal Reports .NET. To search for articles related to Crystal Reports .NET, search on the following keywords: dotnet, .NET, VB .NET, or CSHARP.

Crystal Decisions Technical Papers

For the Crystal product range, this provides a library of technical briefs, release notes, FAQs, how-tos, and so on. To search for technical papers or articles on Crystal Reports .NET, use the checkbox provided on the search page to narrow your results or search for a document name like CRNET*.

Appendix A

This will provide a list of available documents, most of which are in Adobe Acrobat (.pdf) format. It is here you will also find summary lists of items such as articles or sample applications that relate to Crystal Reports .NET.

Crystal Decisions Downloads

Also available from the new search facility, the download site includes a number of downloads of drivers, utilities, and sample applications available for use with Crystal Reports .NET. Sample applications are available that use both Visual Basic .NET and C# code, as well as any software updates and hot fixes that are available.

Crystal Decisions Technical Support

<http://community.crystaldecisions.com/support/answers.asp>

As a registered user of Crystal Reports .NET, you are entitled to use Answers By Email, Crystal Decisions' interactive online support service.

Crystal Developers Journal

www.crystaldevelopersjournal.com

Crystal Developers Journal (CDJ) is an independent publication written to help end users and developers learn advanced techniques for making the most of the tools available from Crystal Decisions, including Crystal Reports, Crystal Analysis, and Crystal Enterprise.

The Web site includes original articles and relevant articles from other Web sites, and you can learn advanced techniques for designing reports and integrating reporting capabilities into your own applications or environment. Learn how to integrate reports with Visual Studio, Visual Studio .NET, Delphi, Powerbuilder, and Java applications with real-world examples and sample code. In addition to frequent updates to the CDJ Web site, they also publish a fortnightly e-mail newsletter full of articles, tips, tricks, and techniques for report design and integration.

Microsoft Newsgroups

Microsoft.public.vb.crystal

Microsoft.public.dotnet.general

Microsoft provides a public newsgroup for using Crystal Reports with VB. You will find a number of postings that relate to older versions of Crystal Reports, but some good information is still contained within the group. To connect to these groups, use Outlook Express, and point your News account to news.microsoft.com, where you will find these groups listed.

MSDN

<http://msdn.microsoft.com/library/en-us/crystlmn/html/crconcristalreports.asp>

MSDN provides a duplication of the documentation found with Crystal Reports .NET and is available in the online MSDN library. You can also find links on MSDN to other resources, including articles from MSDN magazine, other Web sites, and dotnet resources like www.gotdotnet.com.

Sources of Errors

A number of different areas could cause errors with Crystal Reports .NET. In the following sections, we will look at some of the most common ones.

Existing Reports

If you have existing Crystal Reports you have created using a previous version of Crystal Reports (8.5 or below), you can use these reports with Crystal Reports .NET by importing them into your project or leave them as external and reference them within your application. The following shows some of the most common errors that occur with reports and Crystal Reports.NET:

Error	Interpretation
Opening a .NET report in a previous version (8.5 or below) causes a fault.	<p>Once you have saved a report in Crystal Reports .NET, the report cannot be opened in previous versions of Crystal Reports. You should keep a copy of the report if you wish to open it in a previous version.</p> <p>You cannot open Crystal Reports .NET reports in Crystal Reports 8.5 or below because the file format for a .NET report is different from that for previous versions.</p> <p>The Crystal Decisions Knowledge Base recommends two different strategies for using reports you also need to edit outside of Crystal Reports .NET:</p> <p>If you add the report as an Existing item and don't change the report's structures, you should be able to edit the report with the previous version designer (8.5 or below) as the report file remains in its original report format.</p> <p>If you know you need to edit the report outside of Crystal Reports .NET, the best method is to leave the report file alone and reference the file at run time. Rather than importing or creating a new report, you can load the previous version report file into the application at run time and view the reports. If you were to set the report source property with the path and name of the report, the report could then be viewed within your .NET application, but it can still be edited externally with the previous version's Report Designer.</p>

Table continued on following page

Appendix A

Error	Interpretation
Crystal Dictionary is not supported in this version.	Crystal Reports .NET does not support Crystal Dictionaries (or Seagate Info Views) as a data source for reports. If you attempt to import an existing report that uses either of these data sources, you will receive this error message. To utilize this report, you need to either set the location of the report to a valid data source, or you could re-create the report within the Crystal Reports .NET Report Designer.
The formatting of graphs is lost in Crystal Reports .NET.	When importing a report from a previous version of Crystal Reports that includes a graph, the graph formatting you have specified may not translate with the rest of the report. Crystal Reports .NET is missing the Chart Analyzer from the retail version of Crystal Reports, which allows you to use advanced formatting options on your graph. Crystal Reports .NET cannot understand this advanced formatting.
OLAP grids disappear from the report when it is imported into Crystal Reports .NET.	When you attempt to import a report from a previous version of Crystal Reports (Crystal Reports 8.5 or below) that utilizes an OLAP grid, the grid will be dropped when you import your report into Crystal Reports .NET. This happens because Crystal Reports .NET does not support OLAP data at the time of going to press, and no workaround is currently available.
Geographic maps disappear from the report when it is imported into Crystal Reports .NET.	Like OLAP grids, Geographic Mapping is not supported within Crystal Reports .NET. Any existing reports that have a map in them can be used, but the map area will appear blank if you import them into the Crystal Reports .NET designer.

Report Designer

Within the Report Designer itself, a number of areas that can be a problem and sometimes make you think you have done something wrong (even when you haven't!). Like any software product on the market, there are still some issues to work out.

Error	Interpretation
The Field Explorer disappears.	When working in the Crystal Report Designer, you can access the fields that are available for use with your report from the Field Explorer. If you accidentally close this window, you can get it back by selecting View → Other Windows → Document Outline or pressing Ctrl+Alt+T.
Delete button does not function properly in the Field Explorer.	Normally, when you highlight a formula or parameter field in the Field Explorer and click the Delete key, the field will be deleted. With Crystal Reports .NET, this behavior does not work. To delete the field, you will need to highlight it and press the Delete key twice. This behavior has been noted and should be fixed with future releases of the product.

Error	Interpretation
Sort order of fields cannot be set.	When working with record-level sorting in your report, there is no way to set the sort order or precedence other than removing and adding the fields again in order.
The priority of the sort order in the Record Sort Order Control cannot be set directly.	In Crystal Reports .NET, the Record Sort Order control offers no direct method to set the priority order of two or more fields in the Sort Fields list. For example, no up or down buttons allow you to change the list order in the Sort Fields list.

If you need to modify the priority in which fields are sorted, manually remove the fields from the Sort Fields list, and add them again in the desired order.

Database and Data-Related

Error	Interpretation
Problems with string lengths in XML datasets.	When using XML datasets as the data source for your reports, Crystal Reports .NET will treat all of the fields as if they have the maximum length and think they contain 65,000+ characters. If you need to use the length for any of these fields, make sure you use <code>trim</code> in the Crystal Reports formula language prior to applying the record.
Set Location functionality causes errors.	When using the Set Location functionality within Crystal Reports .NET, you can set an existing table location to a new location, for example, pointing a report from a test database to a production database. After you have finished setting the location of your data, the user interface within Crystal Reports will not be updated with this information. This is a known error and should be fixed in future releases.

Subreports

Error	Interpretation
A subreport in Crystal Reports .NET cannot be located.	When editing a subreport within Crystal Reports .NET (by right-clicking the subreport and selecting Edit Subreport), a tab is added to the <i>bottom</i> of the page. Previous versions of Crystal Reports would open a tab at the <i>top</i> of the page. You can use these tabs to navigate between different subreports that exist within your main report.

Table continued on following page

Appendix A

Error	Interpretation
A report with a subreport runs slowly or indefinitely.	<p>With subreports, performance can be a problem if your subreport is processed multiple times. When you create a sub-report and place it on a report, keep in mind that the sub-report's position determines how many times it will be run. For example, a subreport placed in the Report Header will only run once (as the Report Header itself only appears once).</p> <p>A subreport placed in the Details section, however, will run once for each detail record that is shown. In a large report, this can mean that a subreport runs hundreds of times over.</p> <p>If you are using subreports to display information in the Details section of your report, consider creating a SQL command, database view, or stored procedure to provide this information instead of using subreports to display the data.</p> <p>In addition, if you don't need to see the subreport immediately, then consider turning the report into an on-demand subreport, by right-clicking the subreport, selecting Format → Subreport, and checking the appropriate option.</p>

Exporting

Error	Interpretation
Formatting errors occur when exporting to Adobe Acrobat.	<p>When exporting to Adobe Acrobat (.pdf) format, you may encounter a number of formatting errors, including:</p> <ul style="list-style-type: none">Boxes drawn on your report lose their formatting.Double-line borders appear as single-lines.Cross-tab header only appears on first page. <p>These are known errors with exporting to .pdf and should be fixed in future versions of Crystal Reports.</p>

Windows Forms Viewer

Error	Interpretation
Cancel button in the print dialog does not work.	When you preview your report in the Windows Forms Viewer, a print button will allow you to print your report to the printer of your choice. You can print from this dialog with no problems, but the cancel button does not cancel the dialog.

Web Forms Viewer

Error	Interpretation
Drilling down into the group tree cannot be done after using the ShowGroupTree method.	If you use the ShowGroupTree method, a bug in Crystal Reports .NET will not allow you to drill down into the group tree. This issue has been tracked by Crystal Decisions and should be fixed in future releases of the product.
The quality of images in the Web Forms Viewer needs to improve.	When working with reports that contain graphs and other pictures in the Web Form, the default resolution is 96 dpi. This resolution was picked based on a number of factors, including file sizing and download times, but it will often turn graphs and other images a bit grainy.
	A setting in the registry can be changed to alter the magnification ratio for images. For more information on the necessary registry changes, go to the Crystal Decisions Knowledge Base, and search for document number c2010317 for complete instructions.

XML Report Web Services

Error	Interpretation
Web Service ignores the record selection formula.	<p>When working with a report that has been published as an XML Report Web Service, the report's record selection formula cannot start with a commented line, as the Web Service will ignore the rest of the record selection formula and return all available records.</p> <p>The following example shows an incorrect record selection formula:</p> <pre>' This sets the record selection formula {Customer.Country} = "USA"</pre> <p>and the correct version:</p> <pre>{Customer.Country} = "USA"</pre> <pre>' This sets the record selection formula</pre> <p>Comments can appear anywhere after the first line but never on the first line.</p>

Table continued on following page

Appendix A

Error	Interpretation
Access Denied error message encountered.	<p>When working with Server File Reports, reports accessed through the generic report Web Service, you may encounter the following error message: Request Failed with HTTP Status 401: Access Denied.</p> <p>To correct this error, you will need to ensure that the <code>CrystalReportWebFormViewer</code> directory is enabled for anonymous access in IIS. You will also need to restart the WWW Publishing Service before this change will take effect.</p>

B

Migrating Applications to Crystal Reports .NET 2003

If you have existing applications that integrate Crystal Reports .NET from Visual Studio .NET 2003 or a previous version of Visual Studio, you may want to read through this appendix to learn about some of the issues around migrating your application. There are two migration methods covered in this appendix. In the first, we will be looking at upgrading from Visual Studio .NET 2002, and in the second, we will go back a little further and look at migrating a reporting application created in Visual Basic 6 to Visual Studio .NET 2003.

Migration Strategies

Regardless of which version you are migrating from, there are two areas of concern — the first is the actual report files themselves and the second is the code used to view these reports. It is always a good idea to keep a separate copy of the .RPT files used in your application so, if the upgrade process goes awry, you can always add these files back to your application.

For the code that is used to launch your reports, you will be fine when migrating from Visual Studio .NET 2002 to 2003, but applications upgraded from Visual Basic 6 may be problematic. There were a number of different ways you could integrate reports into a VB6 application, and though some of the properties, methods, and events are similar in Crystal Reports .NET, the integration methods are different so you may need to rewrite some code to get your application to work. We'll look at that a little later in this appendix, but we first need to have a look at upgrading from Visual Studio .NET 2002.

Upgrading from Visual Studio .NET 2002

Upgrading to Visual Studio .NET 2003 is a relatively pain-free process. Once you have installed Visual Studio .NET 2003, you can open solutions and projects created using the previous version. You will then be prompted to upgrade the project, which only takes a few seconds. All of the features and functionality from the previous version are supported, and you shouldn't have to change any of your existing code for your application to work.

If you are working with Web applications, some common errors may occur when upgrading. The root cause of the majority of these is the way the References in the project are configured. If you do experience problems or error messages when trying to view reports using the Web viewer, follow these steps to correct the problem:

1. Remove the viewer from your page and in the code view.
2. Remove the line that contains the reference to `CrystalDecisions.Web`.
3. Select Project → References, and verify the 9.15.000 version of the following references is selected:
 - `CrystalDecisions.CrystalReports.Engine`
 - `CrystalDecisions.Reportsource`
 - `CrystalDecisions.Shared`
 - `CrystalDecisions.Web`
 - `CrystalDecisions.Windows.Forms`
4. Add the viewer back to your form, and compile and run the application as normal.

If you still are experiencing problems with your Web or other applications, check the Crystal Decisions Knowledge Base at <http://community.crystaldecisions.com/search/>, and search on the keywords ".NET 2003 UPGRADE" for the latest knowledge base articles concerning upgrade issues. You may also want to apply the latest hotfix to see whether this may correct your problem as well. Hot fixes and other product updates are also available through the search facility. Use the checkbox marked Files and Updates, and search on the keyword ".NET" to find the latest files.

Migrating from Visual Basic 6

The Upgrade Wizard is a tool provided by Microsoft as part of the Visual Studio .NET package to assist us with migrating existing applications to the .NET Framework. Unfortunately, the wizard is more targeted at upgrading the Visual Basic components, which do not include the reports that may have been included in the application.

To invoke the wizard, start Visual Studio .NET, select Open Project, and open an existing Visual Basic 6 application. This will launch the Visual Basic Upgrade Wizard. In this instance, we are going to assume that you have opened an existing Visual Basic 6 project that integrates reports using the Report Designer Component. Go through the wizard steps. Agree that you want to create an .EXE (only option) file, and select your destination folder.

Migrating Applications to Crystal Reports .NET 2003

When the wizard is finished, build and test your report. You should receive an error message concerning the .DSR files that were present in your project as a .DSR file (from the Report Designer Component) must be manually converted to a standard Crystal Report (.RPT) file.

To convert a .DSR file to an .RPT file:

- ❑ From the VB6 project, double-click the .DSR file from the Project window to load the report.
- ❑ Right-click the report, and, from the pop-up menu, select Report → Save to Crystal Reports File.

If we drop the .DSR files from our new Visual Studio .NET 2003 project and insert our converted Crystal files, everything will work to some level of satisfaction. Here is the source code that would have been generated from the resulting .VB file; it contains a to-do list of what you will need to finish upgrading your application:

```
'UPGRADE_ISSUE: CrystalReport1 object was not upgraded. Click for more:  
'ms-help://MS.VSCC/commoner/redir/redirect.htm?keyword="vbup2068"  
Dim Report As New CrystalReport1  
  
Private Sub Form2_Load(ByVal eventSender As System.Object,  
                      ByVal eventArgs As System.EventArgs)  
    Handles MyBase.Load  
    'UPGRADE_WARNING: Screen property Screen.MousePointer has a new behavior.  
    'Click for more:  
    'ms-help://MS.VSCC/commoner/redir/redirect.htm?keyword="vbup2065"  
    System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor  
    'UPGRADE_WARNING: Couldn't resolve default property of object  
    'CRViewer1.ReportSource.  
    'Click for more:  
    'ms-help://MS.VSCC/commoner/redir/redirect.htm?keyword="vbup1037"  
  
    'UPGRADE_WARNING: Couldn't resolve default property of object Report.  
    'Click for more:  
    'ms-help://MS.VSCC/commoner/redir/redirect.htm?keyword="vbup1037"  
  
    CRViewer1.ReportSource = Report  
    CRViewer1.ViewReport()  
    'UPGRADE_WARNING: Screen property Screen.MousePointer has a new behavior.  
    'Click for more:  
    'ms-help://MS.VSCC/commoner/redir/redirect.htm?keyword="vbup2065"  
  
    System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.Default  
End Sub  
  
'UPGRADE_WARNING: Event Form2.Resize may fire when form is initialized.  
'Click for more:  
'ms-help://MS.VSCC/commoner/redir/redirect.htm?keyword="vbup2075"  
Private Sub Form2_Resize(ByVal eventSender As System.Object,  
                      ByVal eventArgs As System.EventArgs)  
    Handles MyBase.Resize  
    CRViewer1.Top = 0  
    CRViewer1.Left = 0  
    CRViewer1.Height = ClientRectangle.Height  
    CRViewer1.Width = ClientRectangle.Width  
End Sub
```

Appendix B

As I mentioned at the start of the appendix, you may want to consider recoding to take care of these errors. Otherwise, you may spend a large amount of time trying to troubleshoot the errors shown, and you still may not end up with a working application. Keep in mind that your form designs will come across nicely. It is only the code behind it that you will need to rewrite or update using the skills you learned earlier in the book.

Again, if you run into problems, visit Crystal Decisions Knowledge Base at <http://community.crystaldecisions.com/search/>, and search on the keywords ".NET 2003 UPGRADE" for the latest knowledge base articles concerning upgrade issues.

C

Crystal Syntax versus Basic Syntax

In Chapter 8, "Formulas and Logic," we looked at the differences between Basic Syntax and Crystal Syntax and created formulas using both. This appendix has been put together as a handy, in-depth reference of the differences between the two and provides listings of the functions and operators in each syntax. This list is by no means exhaustive, but it hopefully provides you with some idea of the main and most-used syntax in cases where Basic and Crystal Syntax differ.

To change between Crystal and Basic Syntax within your report, create or edit an existing formula, and use the drop-down list shown in the upper right corner of Figure C-1 to select the syntax you want to use.

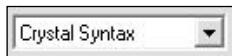


Figure C-1

Remember that you cannot combine both syntaxes in a single formula and that it is possible to use Crystal Syntax only in a record selection formula.

By default, when you open the Formula Editor, the syntax will default to Crystal Syntax, but you can change the default settings through options found in the Report Designer itself. To change these settings, open a report and right-click anywhere within a blank area of your report. From the right-click menu, select Designer → Default Settings and then click the Reporting property page to open the dialog shown in Figure C-2.

Appendix C

You can use the drop-down list provided to change the default syntax that will be selected when you start the Formula Editor. Remember, you can always change this setting manually within the Formula Editor if you would like to use the other syntax.

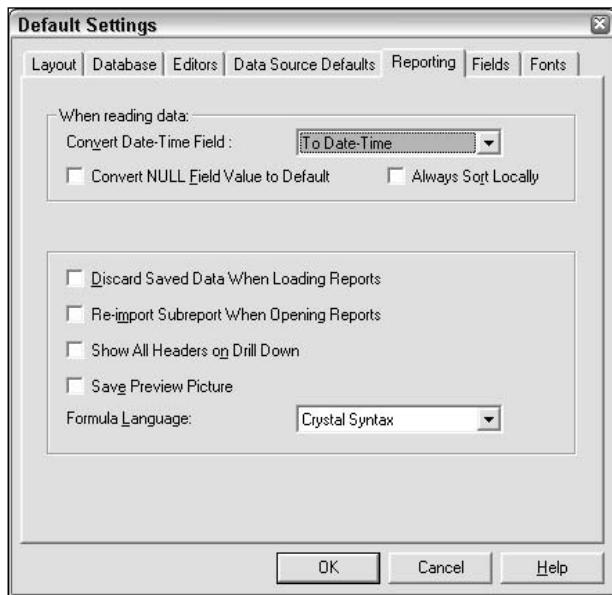


Figure C-2

The following sections detail the differences between Crystal and Basic Syntax—starting with the related functions in each.

Functions

Crystal Reports includes a number of pre-built functions available for use within a formula. The majority of these functions are the same in both Basic and Crystal Syntax with the exception of the following functions.

Where there is more than one equivalent function, they are all listed.

Mathematical Functions

Basic	Crystal
Fix(n)	Truncate(n)
Fix(n, #places)	Truncate(n, #places)

Crystal Syntax versus Basic Syntax

When *n* is a number-type field, both `Fix()` and `Truncate()` will truncate a number to a specified number of decimal places.

For example, `Fix({field}, 2)` would trim the field to two decimal places.

String Functions

Basic	Crystal	Notes
<code>Len(string)</code>	<code>Length(string)</code>	Returns the length of a string
<code>LTrim(string)</code>	<code>TrimLeft(string)</code>	Trims all spaces from the left-hand side of a string
<code>RTrim(string)</code>	<code>TrimRight(string)</code>	Trims all spaces from the right-hand side of a string
<code>UCase (string)</code>	<code>Uppercase (string)</code>	Converts a string to all uppercase characters
<code>LCASE (string)</code>	<code>Lowercase(string)</code>	Converts a string to all lowercase characters
<code>IsNumeric (string)</code>	<code>n/a</code>	Boolean (determines whether a string is numeric or not)
<code>CStr(field), ToText(field)</code>	<code>Cstr(field), ToText(field)</code>	Converts different types of fields to string

Date/Time Functions

Basic	Crystal	Notes
<code>CDate, DateValue</code>	<code>CDate, DateValue</code>	Date
<code>CTime, TimeValue</code>	<code>CTime, TimeValue, Time</code>	Time
<code>CDatetime, DateTimeValue</code>	<code>DateTime, CDateTime, DateTimeValue</code>	Date-Time
<code>WeekDay</code>	<code>DayOfWeek</code>	Returns the day of week

Arrays

Basic	Crystal	Notes
<code>Array (x, ...)</code>	<code>MakeArray (x, ...)</code>	For creating arrays of different types within your formula

Appendix C

Operators

In addition to different types of functions, Basic and Crystal Syntax occasionally utilize different operators:

Arithmetic

Basic	Crystal
n/a	x%y

This is the percent operator, and it calculates the percent that X is of Y.

Basic	Crystal	Notes
&	&, +	Similar in function, the ampersand can be used to concatenate different types of fields whereas the plus operator works only with string fields.
(x(y))	x[y]	For referencing the subscript of arrays and strings, Basic Syntax uses parenthesis while Crystal Syntax uses brackets. (For example {stringfield}[3] would return the character in the third position of the string.)

Variable Declarations

Before we can use a variable in a formula, it has to be declared. Basic and Crystal Syntax both have different ways of declaring variables, based on the type and scope of the variable. The most common variable declarations are included here:

Basic	Crystal
Dim x	n/a
Dim x ()	n/a
Dim x As Boolean	Local BooleanVar x
Dim x As Number	Local NumberVar x
Dim x As Currency	Local CurrencyVar x
Dim x As Date	Local DateVar x
Dim x As Time	Local TimeVar x
Dim x As DateTime	Local DateTimeVar x

Crystal Syntax versus Basic Syntax

Basic	Crystal
Dim x As String	Local StringVar x
Dim x As Number Range	Local NumberVar range x
Dim x As Currency Range	Local CurrencyVar range x
Dim x As Date Range	Local DateVar range x
Dim x As Time Range	Local TimeVar range x
Dim x As DateTime Range	Local DateTimeVar range x
Dim x As String Range	Local StringVar range x
Dim x () As Boolean	Local BooleanVar array x
Dim x () As Number	Local NumberVar array x
Dim x () As Currency	Local CurrencyVar array x
Dim x () As Date	Local DateVar array x
Dim x () As Time	Local TimeVar array x
Dim x () As DateTime	Local DateTimeVar array x
Dim x () As String	Local StringVar array x
Dim x () As Number Range	Local NumberVar range array x
Dim x () As Currency Range	Local CurrencyVar range array x
Dim x () As Date Range	Local DateVar range array x
Dim x () As Time Range	Local TimeVar range array x
Dim x () As DateTime Range	Local DateTimeVar range array x
Dim x () As String Range	Local StringVar range array x

To select a scope for variables created in Basic Syntax, you can use the following scope attributes in place of the `Dim` statement:

- Local—The variable is specific and can be used only in the formula in which it is defined.
- Global—The variable is available to formulas throughout the entire current report.
- Shared—The variable can be shared with a subreport as well as the entire current report.

Dim is equivalent to using “Local.”

Index

SYMBOLS AND NUMERICS

& (**ampersand**), concatenation operator, 261, 342
 @ (**at sign**), prefixing formulas, 251, 256
 \ (**backslash**), integer divide in Basic Syntax, 258
 ^ (**caret**), exponentiation in Basic Syntax, 258
 : (**colon**), preceding REM statement, 257
 {} (**curly brackets**)
 enclosing field names in Basic Syntax, 256
 enclosing parameter field, 56, 101
 = (**equal to operator**)
 Basic Syntax, 259
 joins, 223
 > (**greater than operator**)
 Basic Syntax, 259
 joins, 223
 >= (**greater than or equal to operator**)
 Basic Syntax, 259
 joins, 223
 < (**less than operator**)
 Basic Syntax, 259
 joins, 223
 <= (**less than or equal to operator**)
 Basic Syntax, 259
 joins, 223
 – (**minus sign**), negation operator in Basic Syntax, 258
 <> (**not equal to operator**), Basic Syntax, 259
 != (**not equal to operator**), joins, 223
 () (**parentheses**), subscript operator in Basic Syntax, 342

% (**percent sign**)
 in database object filter, 217
 percentage operator in Crystal Syntax, 342
 preceding SQL expression, 234
 + (**plus sign**), concatenation operator, 261, 342
 # (**pound sign**) preceding running total, 82
 ? (**question mark**) preceding parameter field, 56, 101, 256
 ' (**single quote**), preceding comments in Basic Syntax, 257
 %20, space encoding, 200
 [] (**square brackets**)
 subscript operator in Crystal Syntax, 342
 substring operator in Basic Syntax, 261
 _ (**underscore**), in database object filter, 217
 2-tier applications, 10–11
 2 1/2-tier logic, 246
 3-tier applications, 11–12

A

Access database files, 33, 63, 213
 Acrobat (Adobe)
 exporting reports to, 136, 284
 printing reports from, 186–188
 Add Command To Report window, 229
 Add Modules window, 320
 ADO .NET data driver, 319
 ADO .NET datasets
 browsing data not supported with, 33
 creating, 234–236

ADO .NET datasets

ADO .NET datasets (continued)

creating from database files, 213
creating report from, 239–240
as datasource, 32, 33
modules required for, 15
object model for, 112
support for, 16
viewing contents of, 236–239
viewing reports containing, 240–242

Adobe Acrobat. See Acrobat (Adobe)

Aged0To30Days **function, Basic Syntax**, 263
Aged31To60Days **function, Basic Syntax**, 263
Aged61To90Days **function, Basic Syntax**, 263
alias, database, 226–227
AllDatesFromToday **function, Basic Syntax**, 263
AllDatesFromTomorrow **function, Basic Syntax**, 263
AllDatesToToday **function, Basic Syntax**, 263
AllDatesToYesterday **function, Basic Syntax**, 263
ampersand (&), concatenation operator, 261, 342
AMPMFormat **property, TimeFieldFormat class**, 303
AMString **property, TimeFieldFormat class**, 303
And operator, 258

applications. See also reports

adding Report Engine as reference, 277
adding report to, 30–31, 43–46, 116–117, 159–161
adding Report Web Service as reference, 206–207
architecture of, 9–12
creating, 30, 110–112, 153–154
deploying
 project types for, 312–313
 requirements for, 313–314
 Setup Wizard for, 313
Web-based applications, 322–325
 Windows-based applications, 315–322
migrating to Crystal Reports .NET 2003, 335–338
sample applications, 18–21, 107–108
Web-based applications
 adding report to, 159–161
 creating, 153–154
 deploying, 312, 314, 322–325
 development environment for, 152
 history of, 151–152
 large-scale, 151
 object models for, 154
 parameter fields in, 169–173
 planning, 150–151
 record selection for, 168–169
 server requirements for, 152
 viewing external report in, 156–159
 viewing report in, 162–168

Windows-based applications

 adding report to, 116–117
 creating, 110–112
 database logon information, 121–123
 deploying, 312, 314, 315–322
 object models for, 112
 parameter fields in, 125–130
 planning, 108–109, 112
 record selection for, 124–125
 viewing external report in, 114–115
 viewing report in, 117–121

ApplyLogOnInfo method, TableLogOnInfo class, 286

ApplyPageMargins method, PrintOptions class, 282

architecture of applications, 9–12

AreaFormat class, 295–296

Areas collection, 294

areas of report, controlling, 294–296

arithmetic operators

 for formulas in Basic Syntax, 257–258
 for formulas in Crystal Syntax, 342

Array function, Basic Syntax, 341

Ashwin scheduling product, 9

ASMX files

 for basic Report Web Services, 198
 consuming, 204–208
 creating, 196–202
 for generic Report Web Services, 202–204

ASP .NET, 149, 150. See also Web-based applications;
 XML Report Web Services

assignments, Basic Syntax for, 257

at sign (@), prefixing formulas, 251, 256

author of report, 54, 278

Author property, Setup Project, 316

Average, summary function, 39

B

BackGroundColor property, SectionFormat class, 296

backslash (\), integer divide in Basic Syntax, 258

Basic Syntax for formulas

 array functions, 341
 assignments, 257
 comments in, 257
 control structures, 266–267
 data type conversion functions, 259–260
 data types for, 259
 date and period functions, 262–265

date/time functions, 341
 definition of, 255–256
 differences from Crystal Syntax, 268–269
 field names, 256
 Formula variable, 256–257
 mathematical functions, 340–341
 operators, 342
 operators used for, 257–259
 print state and document functions, 266
 record selection formulas not supported in, 269–270
 selecting, 56, 252
 string functions, 261–262, 341
 summary functions, 260–261
 variable declarations, 342–343
 Visual Basic compared to, 256–257

BestFitPage property, Report Viewer, 175

Binary Large Object (BLOB) data type, Basic Syntax, 259

binding reports to Report Viewer, 118–121, 158, 162–168

BlobFieldObject, 297

books, about Web services, 196. See also resources

Boolean data type, Basic Syntax, 259

Boolean fields
 formatting, 300–301
 parameter fields, 101

Boolean operators, Basic Syntax for, 258

Boolean properties, conditional formatting for, 270–271

BooleanFormat class, 300–301

Border property, FieldObject class, 299

Bottom Percentage analysis, 40, 79–80

BottomN analysis, 40, 79–80

BoxObject, 297

branching statements, Basic Syntax, 266–267

browser
 printing reports from, 184–186
 Report Viewer rendering in, customizing, 178–179

C

cached reports, binding to Report Viewer, 166–168

calculations. See formulas

Calendar1stHalf function, Basic Syntax, 263

Calendar1stQtr function, Basic Syntax, 263

Calendar2ndHalf function, Basic Syntax, 263

Calendar2ndQtr function, Basic Syntax, 263

Calendar3rdQtr function, Basic Syntax, 263

Calendar4thQtr function, Basic Syntax, 263

caret (^), exponentiation in Basic Syntax, 258

Case clause, Basic Syntax, 267

case sensitivity of database server, 217

CBool function, Basic Syntax, 259

CCur function, Basic Syntax, 259

CDate function
 Basic Syntax, 260, 341
 Crystal Syntax, 341

CDatetime function
 Basic Syntax, 260, 341
 Crystal Syntax, 341

CDbL function, Basic Syntax, 259

Change Group Options window, 74–75

Chart Analyzer, not supported, 93

Chart Expert, 89–92

Chart tab, Standard Expert, 40–41

ChartObject, 297

charts
 controlling content in, 92
 features of, 40–41
 formatting, 92
 inserting into report, 89–93
 types of, 89

classes. See also specific classes
 in CrystalDecisions.Web namespace, 155
 in CrystalDecisions.Windows.Forms namespace, 113–114

ClientTarget property, CrystalReportViewerBase class, 178–179

colon (:), preceding REM statement, 257

Color property, FieldObject class, 299

columns. See cross-tabs; tables (database)

ComboBox_Zoom, 138–139

comments
 Basic Syntax for, 257
 in report, 54, 278

comparison operators, Basic Syntax, 258–259

concatenation operator (+ or &) in Basic Syntax, 261, 342

conditional formatting, 270–273

conditional statements, Basic Syntax, 266–267

ConnectionInfo class, 121–123, 168, 286–287

control structures, Basic Syntax, 266–267

Correlation, summary function, 39

Count property, Tables class, 285

Count, summary function, 39

Covariance, summary function, 39

crdb_adoplus.dll file, 15, 319

Create Parameter Field window, 103

Cross-Tab Expert, 31, 83

CrossTabObject, 298

cross-tabs

cross-tabs

charts based on, 91
creating report as, 31, 83
formatting, 86–88
inserting into existing report, 83–85
predefined styles for, 86

Crystal Decisions Downloads, 328

Crystal Decisions Knowledge Base, 327

Crystal Decisions Technical Papers, 327–328

Crystal Decisions Technical Support, 328

Crystal Decisions Web site, 4

Crystal Developers Journal, 22, 328

Crystal Dictionaries, not supported, 14, 214

Crystal Enterprise

application architecture and, 10–12
features of, 6, 8–9, 151
Web site for, 12, 151

Crystal Field Definition (TTX) files, 33, 63

Crystal Queries, not supported, 14, 214

Crystal Report Gallery window, 30, 62

Crystal Report Viewer. See Report Viewer

Crystal Reports Engine object model, 112

Crystal Reports .NET. See also Report Designer; Report Engine; Report Viewer

benefits of, 14–16
Crystal Enterprise and, 6, 8–9, 10–12, 151
features of, 2, 3–14
history of, 1, 3
incompatibilities with retail version, 14
installing, 16–18
licensing for application deployment, 321
new features of, 7–14
registration of, 30, 321
resources for, 21–23
sample applications for, 18–21, 107–108, 110, 150, 328
sample reports for, 4, 20, 26–27, 109, 150
tutorials for, 21–22

Crystal Reports (retail version)

Chart Analyzer, 93
Crystal Dictionaries, 214
Crystal Query, 214
importing reports from, 14
incompatibilities with .NET version, 14
OLAP support, 90
relational database support, 213
Version 10.0 features, 14

Crystal Reports Windows Forms Viewer object model, 112

Crystal Syntax for formulas

array functions, 341
date/time functions, 341
definition of, 56
differences from Basic Syntax, 268–269
mathematical functions, 340–341
operators, 342
record selection formulas in, 269–270
selecting, 252
string functions, 341
variable declarations, 342–343

Crystal_Database_Access2003_enu.MSM file, 15, 319

Crystal_Database_Access2003.MSM file, 15, 319

CrystalDecisions.CrystalReports.Engine namespace, 163, 276

CrystalDecisions.CrystalReports.Engine object model, 154

CrystalDecisions.Shared namespace, 285

CrystalDecisions.Web namespace, 155

CrystalDecisions.Web object model, 154

CrystalDecisions.Windows.Forms namespace, 113–114

Crystal_Managed2003.MSM file, 15, 318

CrystalReportViewer class, 113, 155, 173. See also Report Viewer

CrystalReportViewerBase class, 155

CStr function

Basic Syntax, 260, 341
Crystal Syntax, 341

CTime function

Basic Syntax, 260, 341
Crystal Syntax, 341

curly brackets ({})

enclosing field names in Basic Syntax, 256
enclosing parameter field, 56, 101

Currency data type, Basic Syntax, 259

Currency field type, parameter fields, 101

currency fields, formatting, 305–306

CurrencySymbolFormat property, NumericFieldFormat class, 305

CurrentFieldValue attribute, 273

CurrentGroupLevel property, DrillEventArgs class, 144

CurrentGroupName property, DrillEventArgs class, 144

CurrentGroupPath property, DrillEventArgs class, 144

- CurrentPageNumber **property**, *NavigateEventArgs class*, **141, 189**
- CurrentSubreportName **property**, **145**
- CurrentSubreportPageNumber **property**, **145**
- CurrentSubreportPosition **property**, **145**
- Custom Data Provider**, **213**
- Custom Navigation **sample application**, **20**
- CustomerListing **sample report**, **26**
- CustomerOrders **sample report**, **61**
- ## D
- Data Adapter Configuration Wizard**, **236–237, 240**
- Data Date field**, **53**
- Data tab, Standard Expert**, **32–34**
- Data Time field**, **53**
- data types**
in Basic Syntax, **259**
conversion functions for, **259–260**
for parameter fields, **101**
- Database class**, **285**
- Database Expert window**
adding datasource to report, **62–67, 217–219**
setting database alias, **226–227**
- database fields**, **53**
- database files as datasource**, **33, 63, 212, 213**
- database server**
case sensitivity of, **217**
using to optimize report performance, **105, 217**
- DatabaseName property**, *ConnectionString class*, **122, 286, 287**
- databases. See datasources**
- .DataBind method, Report Viewer**, **158, 162, 170**
- DataDefinition container**, **307**
- Dataset Designer**, **234–235**
- datasets**
ADO .NET datasets
creating, **213, 234–236**
creating report from, **239–240**
modules required for, **15**
viewing contents of, **236–239**
viewing reports containing, **240–242**
pushing data into report from, **291–292**
XML datasets, errors with, **331**
- DataSource property**, *FieldObject class*, **299**
- datasources. See also SQL commands**
access to, methods of, **212–215**
adding to report, **32–34, 64–67**
ADO .NET datasets as, **32, 33, 213, 234–242**
alias for, **226–227**
- changes to, verifying report mappings after, **69, 224–225**
- Custom Data Provider for, **213**
- data retrieval methods for, **13, 291–292**
- database files as, **33, 63, 212, 213**
- database objects
adding to report, **217–221**
selecting, **216–217**
- date and time data read from, **53**
- determining when planning report, **28**
- eliminating duplicates when browsing, **217**
- errors with, **331**
- Excel files as, **33, 63, 213**
- fields from
grouping on, **36–38**
inserting into report, **34–36, 53, 67**
sorting on, **36–38**
- fields not bound to (unbound fields), **58–59**
- filtering (selecting) data
creating record selection formulas, **269–270**
with Report Engine, **293–294**
with Report Expert, **41–43**
for Web-based applications, **168–169**
for Windows-based applications, **124–125**
- formulas in, **246–247**
- location of
changing, **225**
setting, **68–69**
- logon information for, **121–123, 168, 285–287**
- ODBC (RDO) access to, **33, 63, 212, 213, 214**
- OLEDB (ADO) access to, **33, 63, 212, 213, 214**
- optimizing, **104–105, 217**
- project data as, **32, 63, 212, 247**
- pushing data into report, **13, 291–292, 294**
- refreshing report data from, **135, 142, 179, 189–190**
- relational databases as, **213**
- removing from report, **68**
- sample database, **21, 211–212**
- saving data to report, **292**
- SQL commands as, **228–234**
- stored procedures as, **53, 64**
- supported by Crystal Reports .NET, **13, 212–215**
- system tables as, **64**
- tables from
adding to report, **217–221**
inserting into report, **34**
links between, specifying, **65–67, 219, 221–224, 289–291**
location of, setting, **287–289**
logon information for, **121–123, 168**

datasources

datasources (continued)

removing from report, 68
selecting to use in reports, 216
virtual, defining, 228–231
troubleshooting, 331
types of, 32–33, 63
XML database files as, 33, 63

Date data type, Basic Syntax, 259

Date field type, parameter fields, 101

date fields

formatting, 301–303
in Report Designer, 53

date functions, Basic Syntax, 262–265

Date Time field type, parameter fields, 101

DateFieldFormat class, 302–303

DateTime data type, Basic Syntax, 259

date-time fields, formatting, 304

DateTime function, Crystal Syntax, 341

DateTimeFieldFormat class, 304

DateTimeSeparator property

DateTimeFieldFormat class, 304

DateTimeValue function, 341

DateValue function, 341

DayFormat property, DateFieldFormat class, 302

DayOfWeek function, Crystal Syntax, 341

DB/2 databases, 213

dBase/Xbase database files, 213

DecimalPlaces property, NumericFieldFormat class, 305

Default Settings window, 48

DefaultAttribute attribute, 273

deploying applications

licensing requirements for, 321

project types for, 312–313

requirements for, 313–314

Setup Wizard for, 313

Web-based applications, 322–325

Windows-based applications, 315–322

DestinationFields property, TableLink class, 290

DestinationOptions property, ExportOption class, 282

DestinationTable property, TableLink class, 290

Detail area, 294

detail records, drilling down to, 143–145

Details Section, 51

Developers Journal, Crystal, 328

Dim statement, Basic Syntax, 342–343

Direction property, Search event, 142

discrete parameters, 102, 127, 170, 171–172

DisplayBackgroundEdge property, Report Viewer, 131

DisplayGroupTree property, Report Viewer, 131, 175

DisplayPage property, Report Viewer, 175

DisplayToolbar property, Report Viewer, 131, 175

Distinct Count, summary function, 39

distinct data, ensuring when browsing, 217

distributing applications. See **deploying applications**

DLL files, created by XML Report Web Services, 195

DOC files, exporting reports to, 136, 284

Dock property, Report Viewer, 117

document functions, Basic Syntax, 266

Dotnetfx.exe file, 313

Downloads, Crystal Decisions, 328

Drill Down Expert, 32

Drill event, 113, 143–145

DrillDownSubreport event, 113

DrillEventArgs class, 113, 143–144, 155

DrillSubreportEventArgs class, 113, 155

DSN, creating, 26–27

DSR files, converting to RPT files, 337–338

duplicates, eliminating when browsing data, 217

E

EDI, 194

Else clause, Basic Syntax, 267

EnableDrillDown property, 145

EnableHideForDrillDown property, AreaFormat class, 295

EnableKeepTogether property

AreaFormat class, 295

SectionFormat class, 296

EnableNewPageAfter property

AreaFormat class, 295

SectionFormat class, 297

EnableNewPageBefore property

AreaFormat class, 295

SectionFormat class, 297

EnablePrintAtBottomOfPage property

AreaFormat class, 295

SectionFormat class, 297

EnableResetPageNumberAfter property

AreaFormat class, 295

SectionFormat class, 297

EnableSaveDataWithReport property

ReportOptions class, 292

EnableSavePreviewPicture property

ReportOptions class, 292

EnableSaveSummariesWithReport **property**,
ReportOptions **class**, 292

EnableSuppress **property**
AreaFormat class, 295
SectionFormat class, 297

EnableSuppressIfBlank **property**, SectionFormat
class, 297

EnableUnderlaySection **property**, SectionFormat
class, 297

EnableUseDummyData **property**, ReportOptions
class, 292

EnableUseLeadingZero **property**,
NumericFieldFormat **class**, 305

Enter Parameter Values window, 125

enterprise applications. See **Crystal Enterprise**

equal to operator (=)
Basic Syntax, 259
joins, 223

Eqv operator, 258

errors. See also **exception handling**; **troubleshooting**
with datasources, 331
with existing reports, 329–330
when exporting reports, 332
in formulas, 253–255
in Report Designer, 330–331
in subreports, 331–332
in Web Forms Viewer, 333
in Windows Forms Viewer, 332
in XML Report Web Services, 333–334

events in Report Viewer, 113–114, 140–146, 188–191

examples. See **samples**; **tutorials**

Excel (Microsoft)
as datasource, 33, 63, 213
exporting reports to, 136, 284
printing reports from, 188

exception handling, 146. See also **troubleshooting**
Exception property, HandleException event, 146
EventArgs class, 113

Exchange (Microsoft), exporting reports to public fold-
ers of, 284

existing applications, migrating to **Crystal Reports .NET**
2003, 335–338

existing reports, problems with, 329–330

experts. See **report experts**

exponentiation operator (^) in Basic Syntax, 258

Export method, ReportDocument **class**, 187,
282–284

ExportDestinationType property, ExportOption
class, 282

ExportFormatType **property**, ExportOption **class**,
187, 188, 282

exporting reports
errors with, 332
formats supported for, 284
from Report Engine, 282–285
from Report Viewer, 136–137

ExportOption class, 282–284

ExportReport method, Report Viewer, 136–137

F

fat applications. See **single-tier applications**

Feature Examples sample reports, 20

Field Explorer, Report Designer

accessing, 48
field objects in, 52–59
tables in, 219–220

field names, Basic Syntax for, 256

field objects

database fields, 53
formatting, 298–300
formula fields, 54–56, 82–83
Group Name fields, 73–75
parameter fields, 56–58, 94, 101–104, 125–130,
169–173
Running Total fields, 80–83
special fields, 53–54
SQL Expression fields, 58, 232–234, 248–249
from stored procedures, 53, 64
summary fields, 39, 40, 54, 76–80, 91, 292
text objects, 53, 298
types of, 52–59
unbound fields, 58–59

FieldFormat property, FieldObject **class**, 299

FieldObject class, 298, 299–300

fields (database). See also **unbound fields**

grouping on, 36–38
inserting into report, 34–36, 53, 67
sorting on, 36–38

Fields property, Table **class**, 286

fields (report)

customizing at run time, 306–309
resetting running totals based on, 82

Fields tab, Standard Expert, 34–36

File Author field, 54

File Creation Date field, 54

File Path and Name field, 54

filtering database objects, 217

filtering (selecting) data

filtering (selecting) data

creating record selection formulas, 269–270
with Report Engine, 293–294
with Report Expert, 41–43
for Web-based applications, 168–169
for Windows-based applications, 124–125

Fix function, Basic Syntax, 340–341

Font property, FieldObject class, 300

footers

Group Footer, 51, 75, 294
Page Footer, 51, 294
Report Footer, 51, 294

Form Expert, 31

Form Letter Expert, 31

Format Cross-Tab window, 86–87

Format Editor window, 97, 301

FormatOptions property, ExportOption class, 282

formatting

areas of report, 295–296
boolean fields, 300–301
conditional, 270–273
currency fields, 305–306
date fields, 301–303
date-time fields, 304
field objects, 298–300
time fields, 303–304

Form_Load event, 278

forms (printed). See Form Expert

Forms Viewer. See Web Forms Viewer; Windows Forms Viewer

forms (Visual Studio), displaying report in, 43–46, 114–117

forms (Web), displaying report in, 156–162

Formula Editor

appearance of, customizing, 251–252
creating formula fields, 54–56
creating record selection formulas, 269–270, 293–294
default syntax type for, 340

formula fields. See also SQL Expression fields; summary fields

creating, 54–56
resetting running totals based on, 82–83

Formula variable, 256–257, 268

FormulaFieldDefinitions collection, 307–309

formulas. See also running totals; summary fields

Basic Syntax, 255–267
conditional formatting, 270–273
group selection formula, 54
integrating into report, 246–250
record selection formula
creating, 269–270
displaying, 54, 124, 168
editing, 124, 169
SQL expressions as alternative to, 232–234
syntax errors in, 253–255
syntax type
 default, 340
 selecting, 252, 339
when to use, 249–250
writing at run time, 307–309

forums, 23

FoxPro database files, 213

full outer join, 223

G

General Business sample reports, 20

generic Web Report Service, 202–204

geographic mapping, not supported, 14

GetCurrentPageNumber method, Report Viewer, 137–138

Global statement, Basic Syntax, 343

graphs. See charts

greater than operator (>)

Basic Syntax, 259
joins, 223

greater than or equal to operator (>=)

Basic Syntax, 259
joins, 223

Group Footer, 51, 75, 294

Group Header, 51, 75, 294

Group Name fields, 73–75

Group Number field, 54

Group Selection Formula field, 54

Group tab, Standard Expert, 36–38

group tree in Report Viewer, displaying, 131, 175, 177

GroupFooter area, 294

GroupHeader area, 294

grouping

changing groups, 72–73
charts based on, 91
deleting groups, 73
displaying group number, 54
displaying group selection formula, 54
fields used for, 36–38
footers for, 51, 75, 294
formatting groups, 73–75
headers for, 51, 75, 294
inserting groups, 70–72
orphans resulting from, eliminating, 75

performing on server, 105, 217
 purpose of, 69
 resetting running totals based on, 82
 sort orders for, 70
 sorting based on summary fields, 78–80
 specified grouping, 37, 70
GroupNumber function, Basic Syntax, 266
GroupSelection function, Basic Syntax, 266

H

Handled property, viewer events, 140, 189
HandleException event, 113, 146
hardware requirements, for deploying applications, 314
HasDrillUpButton property, Report Viewer, 175
HasGotoPageButton property, Report Viewer, 175
HasPageNavigationButtons property, Report Viewer, 176
HasRefreshButton property, Report Viewer, 176
HasSearchButton property, Report Viewer, 176
HasZoomFactorList property, Report Viewer, 176
headers
 Group Header, 51, 75, 294
 Page Header, 51, 294
 Report Header, 51, 294
Height property, FieldObject class, 300
HourFormat property, TimeFieldFormat class, 303
HourMinuteSeparator property, TimeFieldFormat class, 303
HTML files
 exporting reports to, 284
 printing reports from, 188

I

IDE (Integrated Development Environment), 7
If..Then statement, Basic Syntax
 conditional formatting with, 271–273
 using, 266–267
If..Then..Else statement, Basic Syntax
 conditional formatting with, 273
 using, 267
IIS (Internet Information Server), 152
Imp operator, 258
indexes (database), using to optimize report performance, 105, 217
Informix databases, 213
InitReport event, 279
inner join, 223
Insert Group window, 70–71

Insert Subreport window, 95–96
installation. See also Windows Installer files
 Crystal Reports .NET, 16–18
 .NET Framework, 152
 sample applications, 18–21
Instmsia.exe file, 322
Instmsiw.exe file, 322
InStr function, Basic Syntax, 262
integer divide operator (\) in Basic Syntax, 258
Integrated Development Environment. See IDE integration methods for reports, 8. *See also Web Forms Viewer; Windows Forms Viewer; XML Report Web Services*
Interactivity sample application, 20
Internet Explorer (Microsoft), specifying for Report Viewer, 178
Internet Information Server. See IIS
IsNull function, Basic Syntax, 266
IsNumeric function, Basic Syntax, 262, 341
Item property, Tables class, 285

J

joins
 creating in Database Expert, 65–67, 221–224
 looping through, 289–291
 in views, 246
JoinType property, TableLink class, 290

K

KeywordsInReport property, SummaryInfo class, 278
Knowledge Base, Crystal Decisions, 327

L

languages
 for merge modules, other than English, 319
 programming, supported by Visual Basic .NET, 7
Last7Days function, Basic Syntax, 262
LastFullMonth function, Basic Syntax, 263
LastFullWeek function, Basic Syntax, 262
Last4WeeksToSun function, Basic Syntax, 262
LastYearMTD function, Basic Syntax, 263
LastYearYTD function, Basic Syntax, 263
LCase function, Basic Syntax, 262, 341
left outer join, 223
Left property, FieldObject class, 300
Len function, Basic Syntax, 262, 341

Length function, Crystal Syntax

Length function, Crystal Syntax, 341

less than operator (<)

Basic Syntax, 259

joins, 223

less than or equal to operator (<=)

Basic Syntax, 259

joins, 223

letters. See Form Letter Expert

LicenseKey property, 15

Licensing for application deployment, 321

LineObject, 298

linked subreports

changing links for, 97

creating links for, 96

definition of, 94

links between tables

creating in Database Expert, 65–67, 219, 221–224

looping through, 289–291

Links class, 285

Load event, 142–143

Load method, 118

Local statement

Basic Syntax, 343

Crystal Syntax, 342–343

Location property, Table class, 286, 288

logon information for database, 121–123, 168, 285–287

LogOnInfo property, Table class, 286

Lowercase function, Crystal Syntax, 341

LTrim function, Basic Syntax, 262, 341

M

Mail Label Expert, 32

Mail (Microsoft), exporting reports to, 284

mailing labels. See Mail Label Expert

MakeArray function, Crystal Syntax, 341

Manufacturer property, Setup Project, 316

MapObject, 298

mathematical functions, 340–341

Maximum, summary function, 39

Median, summary function, 39

merge modules

adding to Setup Projects, 319–321

for ADO .NET datasets, 319

definition of, 313

for languages other than English, 319

list of, 15, 318–319

methods. See also specific methods

for Report Viewer, 134–139, 179–182

for XML Report Web Services, listing, 204

Microsoft Excel. See Excel (Microsoft), exporting reports to

Microsoft Newsgroups, 328

Microsoft Word. See Word (Microsoft), exporting reports to

migrating applications to Crystal Reports .NET 2003, 335–338

Minimum, summary function, 39

minus sign (-), negation operator in Basic Syntax, 258

MinuteFormat property, TimeFieldFormat class, 303

MinuteSecondSeparator property, TimeFieldFormat class, 303

Mod operator, in Basic Syntax, 258

Mode, summary function, 39

Modification Date field, 53

Modification Time field, 53

modulus operator in Basic Syntax, 258

MonthFormat property, DateFieldFormat class, 302

MonthToDate function, Basic Syntax, 262

MSCVR71.dll file, 319

MSDN Web site, 328–329

MSI files, 312. See also Windows Installer files

MsiNetAssemblySupport condition, 314

MSM files, 15

MSVCP71.dll file, 319

multi-tier applications, 12

N

Name property

FieldObject class, 300

Table class, 286

namespaces

CrystalDecisions.CrystalReports.Engine namespace, 163, 276

CrystalDecisions.Shared namespace, 285

CrystalDecisions.Web, 155

CrystalDecisions.Windows.Forms, 113–114

native connection to datasource, 212

Navigate event, 113, 141, 189

NavigateEventArgs class, 113, 141, 155, 189

negation operator (-) in Basic Syntax, 258

NegativeFormat property, NumericFieldFormat class, 305

.NET Framework

Crystal Reports and, 7–14

installing, 152

required for application deployment, 313, 323

required for XML Report Web Services, 209

PageMargins property, PrintOptions class

NewGroupLevel **property**, DrillEventArgs **class**, **144**
 NewGroupName **property**, DrillEventArgs **class**, **144**
 NewGroupPath **property**, DrillEventArgs **class**, **144**
 NewPageNumber **property**, NavigateEventArgs **class**, **141, 189**
newsgroups, **23, 328**
 NewSubreportName **property**, **145**
 NewSubreportPageNumber **property**, **145**
 NewSubreportPosition **property**, **145**
 NewZoomFactor **property**, ZoomEventArgs **class**, **143, 190**
 Next30Days **function**, **Basic Syntax**, **263**
 Next31To60Days **function**, **Basic Syntax**, **263**
 Next61To90Days **function**, **Basic Syntax**, **263**
 Next91To365Days **function**, **Basic Syntax**, **263**
 NextIsNull **function**, **Basic Syntax**, **266**
 NextValue **function**, **Basic Syntax**, **266**
not equal to operator (<>), Basic Syntax, **259**
not equal to operator (!=), joins, **223**
Not operator, **258**
Nth Largest, **summary function**, **39**
Nth Most Frequent, **summary function**, **39**
Nth Percentile, **summary function**, **39**
Nth Smallest, **summary function**, **39**
Number data type, **Basic Syntax**, **259**
Number field type, parameter fields, **101**
 NumericFieldFormat **class**, **305**

O

object models

for Web-based applications, **154**
 for Windows-based applications, **112**
 ObjectFormat **property**, FieldObject **class**, **300**
objects
 database objects, **216–221**
 field objects
 database fields, **53**
 formatting, **298–300**
 formula fields, **54–56, 82–83**
 Group Name fields, **73–75**
 parameter fields, **56–58, 94, 101–104, 125–130, 169–173**
 Running Total fields, **80–83**
 special fields, **53–54**
 SQL Expression fields, **58, 232–234, 248–249**
 from stored procedures, **53, 64**
 summary fields, **39, 40, 54, 76–80, 91, 292**
 text objects, **53, 298**

types of, **52–59**
 unbound fields, **58–59**
 report objects
 binding report by, **118–119, 163–164**
 types of, **297–298**
 text objects, **53**
ODBC datasource Administrator, creating DSN, **26–27**
ODBC (RDO) access to datasource, **33, 63, 212, 213, 214**
OLAP (Online Analytical Data Processing), not supported, **14, 214**
OlapGridObject, **298**
OLEDB (ADO) access to datasource, **33, 63, 212, 213, 214**
OLEDB for OLAP, **214**
OleDbDataAdapter, **236, 240**
on-demand subreports, **97–99**
OnFirstRecord function, **Basic Syntax**, **266**
OnLastRecord function, **Basic Syntax**, **266**
Online Analytical Data Processing (OLAP), not supported, **14**
operating system requirements, for deploying applications, **314**
operators
 for formulas in Basic Syntax, **257–259**
 for joins, **223**
Or operator, **258**
Oracle databases, **213**
outer joins, **223**
OutputType property, BooleanFieldFormat **class**, **300–301**
Over90Days function, **Basic Syntax**, **263**

P

Packaging and Deployment Wizard, **311–312**
Page Footer, **51, 294**
Page Header, **51, 294**
Page N of M field, **54**
Page Number field, **54**
Page Setup window, **49**
PageContentHeight property, PrintOptions **class**, **281**
PageContentWidth property, PrintOptions **class**, **281**
PageFooter area, **294**
PageHeader area, **294**
Page_Init event, **158, 162, 163–164, 170**
Page_Load event, **178, 184**
PageMargins property, PrintOptions **class**, **281**

PageNofM function, Basic Syntax

PageNofM function, Basic Syntax, **266**

PageNumber function, Basic Syntax, **266**

PageNumberToBeginSearch property, Search event, **142**

pages

- breaking
 - between areas, **295**
 - between groups, **75**
- displaying in Web-based applications, **175**
- headers and footers for, **51**, **294**
- navigating in Report Viewer, **137–138**, **179–180**
- navigation events in Report Viewer, **141**, **189**
- numbering
 - in cross-tabs, **84**
 - fields for, **53**, **54**
- repeating group headers on, **75**

PageToTreeRatio property, Report Viewer, **177**

PageZoomFactor property, Report Viewer, **177**

PaperOrientation property, PrintOptions class, **281**

PaperSize property, PrintOptions class, **281**

PaperSource property, PrintOptions class, **281**

Paradox database files, **213**

parameter fields

- creating, **56–58**, **101–104**
- customizing user interface for, **126–129**, **169–173**
- data types for, **101**
- definition of, **101**
- in linked subreports, **94**
- setting value of, **103**
- types of values for, **102**, **170**
- in Web-based applications, **169–173**
- in Windows-based applications, **125–130**

ParameterFieldInfo property, Report Viewer, **127**, **171**

ParameterFields collection, **127**

parentheses (()), subscript operator in Basic Syntax, **342**

Password property, ConnectionInfo class, **122**, **286**

PDF files

- exporting reports to, **136**, **284**
- printing reports to, **186–188**

percent sign (%)

- in database object filter, **217**
- percentage operator in Crystal Syntax, **342**
- preceding SQL expression, **234**

performance, optimizing, **104–105**, **217**

period functions, Basic Syntax, **262–265**

PictureObject, **298**

platforms. See also XML Report Web Services

- required for deploying applications, **313–314**
- supported by Crystal Enterprise, **8**

plus sign (+), concatenation operator, **261**, **342**

PMString property, TimeFieldFormat class, **303**

Population Standard Deviation, summary function, **39**

Population Variance, summary function, **39**

pound sign (#) preceding running total, **82**

PreviousIsNull function, Basic Syntax, **266**

PreviousValue function, Basic Syntax, **266**

Print Date field, **53**

print state functions, Basic Syntax, **266**

Print Time field, **53**

Printer Setup window, **49–50**

PrinterDuplex property, PrintOptions class, **281**

PrinterName property, PrintOptions class, **281**

printing reports

- from Report Engine, **279–282**
- from Report Viewer, **134–135**, **182–188**

PrintOptions class, **280–282**

PrintReport method, Report Viewer, **134–135**

PrintToPrinter method, ReportDocument class, **279–280**

problems. See troubleshooting

ProductName property, Setup Project, **316**

Professional ASP .NET 1.0, **196**

programming languages, supported by Visual Basic

- .NET, **7**

project data

- as datasource, **32**, **63**, **212**, **247**
- formulas and logic in, **247**

projects. See applications

properties. See also specific properties

- Boolean, conditional formatting for, **270–271**
- multiple-outcome, conditional formatting for, **271–273**

Properties window, Report Viewer, **130**

prototype of report, **29**

Proxy Method, of adding Web services to application, **206–207**

publications, about Web services, **196**. *See also resources*

pull mode of data retrieval, **13**, **291**

push mode of data retrieval. See also XML Report Web Services

- definition of, **13**
- with Report Engine, **291–292**
- with SQL commands, **294**

Q

- queries.** See **Crystal Queries; SQL commands**
Query Builder, 237–238
question mark (?) preceding parameter field, 56, 101, 256
quote, single ('), preceding comments in Basic Syntax, 257

R

- ranged parameters**, 102, 127, 170, 173
Record Number field, 54
record selection
 creating formulas for, 269–270
 with Report Engine, 293–294
 with Report Expert, 41–43
 for Web-based applications, 168–169
 for Windows-based applications, 124–125
Record Selection Formula field, 54
Record Sort Order window, 76
record-level sorting, 75–76
RecordNumber function, Basic Syntax, 266
RecordSelection function, Basic Syntax, 266
refresh events in Report Viewer, 142, 189–190
Refresh method, 289
RefreshReport method, Report Viewer, 135, 179
registration, of Crystal Reports .NET, 30, 321
REGWIZ.MSM file, 15
Regwiz.msm file, 319, 321
relational databases, 213
REM statement, 257
Report Comments field, 54
Report Designer
 adding fields to report, 52–59
 browsing data not supported with ADO .NET data-source, 33
 charts, adding to report, 89–93
 cross-tabs, adding to report, 83–88
 datasource access through, 215
 datasource, specifying for report, 64–69
 default properties, setting, 48–50
 errors with, 330–331
 features of, 3–4, 12–13
 Field Explorer in, 48, 52–59, 219–220
 grouping records, 69–75
 IDE and, 7
 parameter fields, adding to report, 101–104
 report sections in, 50–52
 running totals, adding to report, 80–83

- sorting groups, 78–80
 sorting records, 75–76
 subreports, adding to report, 93–100
 summary fields, adding to report, 76–78
 troubleshooting, 330–331
 user interface for, 47–48

Report Engine

- adding to application as a reference, 277
 areas of report, controlling, 294–296
 as COM+ object, 8
 database options, setting for report, 292–293
 datasource, logging on to, 285–287
 exporting reports, 282–285
 features of, 4
 fields in report
 customizing at run time, 306–309
 formatting, 298–306
 initializing report, 279
 links between tables, specifying, 289–291
 loading report, 278
 namespace for, 163, 276
 printing reports, 279–282
 pushing data into report, 291–292
 record selection, 293–294
 report objects, controlling, 297–306
 ReportDocument class and, 277
 sections of report, controlling, 296–297
 table location, setting, 287–289

Report Engine object model. See Crystal Reports Engine object model**report experts**

- Chart Expert, 89–92
 choosing as method of report creation, 31
 Cross-Tab Expert, 31, 83
 Database Expert, 62–67, 217–219, 226–227
 Drill Down Expert, 32
 Form Expert, 31
 Form Letter Expert, 31
 Mail Label Expert, 32
 Section Expert, 50–52, 75
 Select Expert, 124
 Standard Expert, 31–43, 228–231
 Subreport Expert, 31, 95
 Top N Expert, 79
 types of, 31–32

Report Footer, 51**Report Header**, 51, 294**report objects**

- binding report by, 118–119, 163–164
 types of, 297–298

Report Options window

- Report Options window, 49**
- Report Requirements document, 28–29**
- Report Title field, 54**
- Report Viewer**
- binding Report Web Service to, 207
 - features of, 15
 - integrating with applications, 8
 - viewing external report in, 44–46
 - viewing external Report Web Services in, 205–206
 - viewing reports containing ADO .NET datasets, 240–242
 - Web-based applications
 - adding to Web form, 162
 - appearance of, customizing, 174–177
 - behavior of, customizing, 177–188
 - binding report to, 158, 162–168
 - browser rendering of, customizing, 178–179
 - errors with, 333
 - events in, 188–191
 - group tree, displaying, 175
 - namespace for, 155
 - object models for, 154
 - page navigation in, 179–180
 - printing a report from, 182–188
 - refreshing data from, 179, 189–190
 - searching report in, 182, 190
 - toolbar, displaying, 175–176
 - viewing external report in, 156–159
 - zooming in, 190
 - Windows-based applications
 - adding to a form, 114–115, 117–118
 - appearance of, customizing, 130–132
 - behavior of, customizing, 132–140
 - binding report to, 118–121
 - drilling down to detail records, 143–145
 - errors with, 332
 - events in, 140–146
 - exception handling in, 146
 - exporting reports from, 136–137
 - group tree, displaying, 131
 - integrating with applications, 109
 - namespace for, 113–114
 - object models for, 112
 - page navigation in, 137–138, 141
 - printing a report from, 134–135
 - refreshing data from, 135, 142
 - report background edge, displaying, 131
 - searching report in, 139–140, 142
 - toolbar, displaying, 131
 - viewing external report in, 114–115
 - zooming in, 138–139, 143

- Report Viewer object model. See Crystal Reports Windows Forms Viewer object model**
- Report Web Services. See XML Report Web Services**
- ReportAuthor property, SummaryInfo class, 278**
- ReportComments property, SummaryInfo class, 278**
- ReportDefinition class, 294, 298**
- ReportDocument class. See also Report Engine**
- creating report object as, 116, 119, 163, 278
 - strongly typed, 121, 166
 - untyped, 120, 164–165
- ReportFooter area, 294**
- ReportHeader area, 294**
- ReportOptions class, 292–293**
- ReportPath property, 203**
- ReportRefresh event, 142, 189–190**
- reports. See also applications; datasources; Report Designer; Report Engine; Report Viewer**
- adding to application, 43–46, 114–117, 159–161
 - analysis features for, 40, 78–80
 - areas of, 294–296
 - author of, 54, 278
 - binding to Report Viewer, 118–121, 158, 162–168
 - charts in, 40–41, 89–93
 - comments in, 54, 278
 - creating from ADO .NET datasets, 239–240
 - creating from scratch, 62
 - creating with Standard Expert, 31–46
 - creation date of, 54
 - creation methods for, 31–32
 - data date and time for, 53
 - data objects for
 - adding, 217–221
 - selecting, 216–217
 - default properties for, setting, 48–50
 - drilling down to detail records, 143–145
 - existing, problems with, 329–330
 - exporting to another format, 136–137, 282–285, 332
 - feasibility of, determining, 29
 - field objects in, types of, 52–59
 - file path and name of, 54
 - filtering (selecting) data for, 41–43, 124–125, 168–169, 269–270, 293–294
 - formatting and layout options, 48–50
 - formatting at object level, 52
 - formulas, adding to, 246–250
 - graphs in, 40–41, 89–93
 - headers and footers for, 51, 75, 294
 - integration methods for, 8
 - modification date and time for, 53
 - name of, binding report by, 118, 162
 - navigating in Report Viewer, 137–138, 141, 179–180

SeparatePages property, Report Viewer

performance of, optimizing, 104–105, 217
 planning, 27–29
 previewing, 44–46
 print date and time for, 53
 printing, 134–135, 182–188, 279–282
 prompting user for information when running, 56–58
 prototype of, 29
 referencing externally from application, 114–115, 118–119, 156–159
 refreshing report data for, 135, 142, 179, 189–190
 requirements for, determining, 28
 sample reports, 4, 20, 26–27, 93, 109, 150
 searching in Report Viewer, 139–140, 142, 182
 sections of, 50–52, 296–297
 Server Reports, 202–204
 strongly typed reports, 121, 166–168
 style for, 43
 subject of, 278
 subreports
 border of, 97
 drilling down to, 145–146
 errors with, 331–332
 inserting into report, 94–96
 linked, 94, 96, 97
 name of, 96
 on-demand, 97–99
 reimporting, 100
 saving independently of main report, 100
 types of, 93–94
 summaries in, 38–39
 technical review for, 28–29
 text in, 53
 thumbnail picture of, saving, 292
 title of, 43, 54, 278
 untyped reports, 120–121, 164–165
 viewing in Web-based application, 156–159, 162–168
 viewing in Windows-based application, 114–115, 117–121
 zooming in Report Viewer, 138–139, 143

ReportSource property
 setting for external reports in Report Viewer, 114, 156–157
 setting for external reports in Report Web Service, 206
 setting SQL command as, 231
 setting to bind by report name, 118, 162

ReportSubject property, SummaryInfo class, 278

ReportTitle property, SummaryInfo class, 278

resources, 22–23, 327–329

Rich Text Format (RTF)
 exporting reports to, 136, 284
 printing reports to, 188

right outer join, 223

RoundingFormat property, NumericFieldFormat class, 305

rows. See cross-tabs; tables (database)

RPT files
 converting DSR files to, 337–338
 definition of, 3
 exporting reports to, 284

RTF files
 exporting reports to, 136, 284
 printing reports to, 188

RTrim function, Basic Syntax, 262, 341

running totals, 80–83. See also formulas; summary fields

S

Sample Standard Deviation, summary function, 39

Sample Variance, summary function, 39

sample Web service, 195

samples. See also tutorials
 applications
 from Crystal Decisions Downloads, 328
 installing, 18–21
 Web-based applications, 150
 Windows-based applications, 107–108, 110
 databases, 21, 211–212
 reports
 chart as subreport, 93
 location of, 20, 109, 150
 for report design, 26–27
 Web site for, 4
 Web services, 193–194, 195

Seagate Info SDK, 9–10

Seagate Info Views, not supported, 14

Search event, 113, 142, 190

search events in Report Viewer, 190

SearchEventArgs class, 113, 155

SearchForText method, Report Viewer, 139, 182

searching a report, 139–140, 142, 182, 190

SecondFormat property, TimeFieldFormat class, 303

Section Expert window, 50–52, 75

SectionFormat class, 296–297

sections of report, controlling, 50–52, 296–297

Select Expert, 124

Select statement, Basic Syntax, 267

Select tab, Standard Expert, 41–43

selecting data. See filtering (selecting) data

SelectionFormula property, 125, 168–169

SeparatePages property, Report Viewer, 175, 184

server (database)

server (database)

case sensitivity of, 217
using to optimize report performance, 105, 217

Server Explorer, 202, 207–208

Server Reports, 202–204

server (Web)

copying Report Web Services to, 208–209
preparing for application deployment, 323
requirements for Web-based applications, 152
virtual directory on, 153–154
`ServerFileReportManager.asmx file`, 204
`ServerFileReportService.asmx file`, 202–203
`ServerName` property, `ConnectionInfo` class, 122, 286, 287

Set Location window, 68

Setup Projects

building, 322
creating, 315–317
definition of, 312
deploying, 322
merge modules for, 15, 319–321
properties of, 316
run-time file requirements for, 318–319
selecting outputs for, 317–318

Setup Wizard, 313

`setup.exe` file, 322, 324
`setup.ini` file, 322

Shared statement, Basic Syntax, 343

`ShowCloseButton` property, Report viewer, 131
`ShowExportButton` property, Report viewer, 131
`ShowFirstPage` method, Report Viewer, 137, 180
`ShowGotoPageButton` property, Report viewer, 131
`ShowGroupTreeButton` property, Report viewer, 131
`ShowLastPage` method, Report Viewer, 137, 180
`ShowNextPage` method, Report Viewer, 137, 180
`ShowNthPage` method, Report Viewer, 138
`ShowPageNavigateButtons` property, Report viewer, 131

`ShowPreviousPage` method, Report Viewer, 137, 180

`ShowPrintButton` property, Report viewer, 131

`ShowRefreshButton` property, Report viewer, 131

`ShowTextSearchButton` property, Report viewer, 131

`ShowZoomButton` property, Report viewer, 131

Simple Page sample application, 20

single quote ('), preceding comments in Basic Syntax, 257

single-tier applications, 10

Solution Explorer

merge modules in, 319
new report shown in, 160–161

Sort All analysis, 40, 79–80

sorting

fields used for, 36–38, 76
record-level sorting, 75–76
by summary fields, 40, 78–80
`SourceFields` property, `TableLink` class, 290

`SourceTable` property, `TableLink` class, 290

spaces, in XML Report Web Services names, 200

special fields, 53–54

specified grouping, 37, 70

spreadsheets. See cross-tabs

SQL commands. See also record selection

creating ADO .NET datasets with, 237–238, 240–241
creating virtual tables with, 229–230
as datasource, 228–231
formulas and logic in, 247–248
generated by Crystal Reports
displaying, 224
optimizing, 104–105
record selection affecting, 294
troubleshooting, 231–232

SQL Expression Editor, 232–233

SQL Expression fields

creating, 58, 232–234
formulas and logic in, 248–249

SQL Expression Name window, 232

SQL*Plus Query Analyzer, 105

SQL Server databases, 213

square brackets ([])

subscript operator in Crystal Syntax, 342
substring operator in Basic Syntax, 261

Standard Expert

Chart tab, 40–41
Data tab, 32–34
definition of, 29, 31
Fields tab, 34–36
Group tab, 36–38
Select tab, 41–43
Style tab, 43
TopN tab, 40
Total tab, 38–39
virtual tables, defining, 228–231

statements. See Form Letter Expert; SQL commands

stored procedures

formulas and logic in, 246–247
selecting to use in reports, 216

stored procedures as datasource, 53, 64

String data type, Basic Syntax, 259

String field type, parameter fields, 101

string functions, Basic Syntax, 261–262

strongly typed cached reports, binding to Report Viewer, 166–168
strongly typed reports, binding to Report Viewer, 121, 166
StrReverse function, Basic Syntax, 262
Style tab, Standard Expert, 43
subject of report, 278
Subreport Expert
 creating subreports with, 95
 definition of, 31
SubreportObject, 298
subreports
 border of, 97
 creating, 95
 drilling down to, 145–146
 errors with, 331–332
 inserting into report, 94–96
 linked, 94, 96, 97
 name of, 96
 on-demand, 97–99
 reimporting, 100
 saving independently of main report, 100
 types of, 93–94
subscript operator, 342
substring operator ([] in Basic Syntax, 261
Sum, summary function, 39
summary fields. *See also formulas; running totals*
 analysis features for, 40, 78–80
 changing, 78
 charts based on, 91
 inserting, 54, 76–77
 operators for, 39
 saving with report, 292
 sorting groups based on, 78–80
summary functions, Basic Syntax, 260–261
Summary Information, comments in, 54
summary report. *See Drill Down Expert*
SummaryInfo class, 278–279
support. *See resources; troubleshooting*
Suppress property, 270
Sybase databases, 213
synonyms, selecting to use in reports, 216
system requirements, for deploying applications, 313–314
system tables
 as datasource, 64
 selecting to use in reports, 216

T

Table class, 286
TableLink class, 289–291
TableLogOnInfo class, 286
TableLogonInfo collection, 121–123
Tables class, 285–286
tables (database)
 adding to report, 34, 217–221
 links between, specifying, 65–67, 219, 221–224, 289–291
 location of, setting, 287–289
 logon information for, 121–123, 168
 removing from report, 68
 selecting to use in reports, 216
 virtual, defining, 228–231
tables (spreadsheet). See cross-tabs; Excel (Microsoft)
Technical Papers, Crystal Decisions, 327–328
technical review for report, 28–29, 247
technical support. *See resources; troubleshooting*
Technical Support, Crystal Decisions, 328
text objects, 53, 298
TextObject, 298
TextToSearch property, Search event, 142
three-tier applications, 11–12
thumbnail picture of report, 292
Time data type, Basic Syntax, 259
Time field type, parameter fields, 101
time fields, formatting, 303–304
Time function, Crystal Syntax, 341
TimeBase property, TimeFieldFormat class, 303
TimeFieldFormat class, 303–304
TimeValue function, 341
title of report, 43, 54, 278
ToNumber function, Basic Syntax, 260
toolbar in Report Viewer, displaying, 131, 175–176
Top Percentage analysis, 40, 79–80
Top property, FieldObject class, 300
TopN analysis, 40, 79–80
TopN Expert window, 79
TopN tab, Standard Expert, 40
Total Page Count field, 54
Total tab, Standard Expert, 38–39
TotalPageCount function, Basic Syntax, 266
totals, running, 80–83. *See also summary fields*
ToText function
 Basic Syntax, 260, 341
 Crystal Syntax, 341

ToWords function, Basic Syntax

ToWords function, Basic Syntax, **260**
Trim function, Basic Syntax, **262**
TrimLeft function, Crystal Syntax, **341**
TrimRight function, Crystal Syntax, **341**
troubleshooting. *See also exception handling*
binding report to Report Viewer, problems with, **121**
datasources, problems with, **331**
error involving keycodev2.dll, **15**
existing reports, problems with, **329–330**
exporting reports, problems with, **332**
Report Designer, problems with, **330–331**
resources for, **327–329**
SQL commands, errors resulting from, **231–232**
subreports, problems with, **331–332**
Web Forms Viewer, problems with, **333**
Windows Forms Viewer, problems with, **332**
XML Report Web Services, problems with, **333–334**
Truncate function, Crystal Syntax, **340–341**
TTX files. *See Crystal Field Definition (TTX) files*
tutorials, **21–22**
%20, space encoding, **200**
2 1/2 tier logic, **11–12**
two-tier applications, **10–11**
type conversion functions, Basic Syntax, **259–260**

U

UCase function, Basic Syntax, **262, 341**
unbound fields, **58–59**
underscore (_), in database object filter, **217**
unlinked subreports, **93**
untyped reports, binding to Report Viewer, **120–121, 164–165**
Upgrade Wizard, **336–337**
upgrading applications. *See migrating applications to Crystal Reports .NET 2003*
Uppercase function, Crystal Syntax, **341**
UserData property, HandleException event, **146**
UserID property, ConnectionInfo class, **122, 286**

V

variables
declarations, **342–343**
in linked subreports, **94**
VC_CRT module, **15**
VC_STL module, **15**
VC_User_CRT71_RTL_X86.msm file, **319**
VC_User_STL71_RTL_X86.msm file, **319**

verifying report mappings, **69, 224–225**
Version property, Setup Project, **316**
Viewer events, **113, 142–143**
ViewerDemo sample application, **26, 61**
ViewerEventArgs class, **113, 155**
views
formulas and logic in, **246–247, 248**
selecting to use in reports, **216**
ViewZoom event, **114, 143, 190**
virtual directory, **153–154**
virtual tables, defining, **228–231**
Visual Basic
compared to Basic Syntax, **256–257**
migrating from, **336–338**
Visual Basic .NET, languages supported by, **7**
Visual Linking Expert, **221–224**
Visual Studio .NET
creating application with, **30, 110–112**
integration with Crystal Reports .NET, **15, 109–110**
version 2002, upgrading from, **336**

W

Web Forms Viewer. *See also Report Viewer*

errors with, **333**
features of, **4, 15**
namespace for, **155**
object models for, **154**
Web server
copying Report Web Services to, **208–209**
preparing for application deployment, **323**
requirements for Web-based applications, **152**
virtual directory on, **153–154**

Web Service Description Language. *See WSDL*

Web Services. *See XML Report Web Services*

Web Setup Projects, **209, 312, 323–325**

Web sites

Crystal Decisions, **4**
Crystal Decisions Knowledge Base, **327**
Crystal Decisions Technical Support, **328**
Crystal Developers Journal, **328**
Crystal Developers Journal Web site, **22**
Crystal Enterprise, **12, 151**
Crystal Reports 10.0, **14**
Microsoft newsgroups, **23**
Microsoft Newsgroups, **328**
MSDN, **328–329**
sample reports, **4**
tutorials, **21**

Web-based applications

- adding report to, 159–161
- creating, 153–154
- deploying, 312, 314, 322–325
- development environment for, 152
- history of, 151–152
- large-scale, 151
- object models for, 154
- parameter fields in, 169–173
- planning, 150–151
- record selection for, 168–169
- server requirements for, 152
- viewing external report in, 156–159
- viewing report in, 162–168

WebForms.exe file, 19**WeekDay function, Basic function, 341****WeekToDateFromSun function, Basic Syntax, 262****Weighted Average, summary function, 39****Width property, FieldObject class, 300****Windows Forms Viewer. See also Report Viewer**

- errors with, 332
- features of, 4, 15
- namespace for, 113–114
- object models for, 112

Windows Installer files

- creating with Setup Projects, 312
- creating with Web Setup Projects, 312
- definition of, 312
- for Report Web Service, 209
- for Setup Project, 322

Windows-based applications

- adding report to, 116–117
- creating, 110–112
- database logon information, 121–123
- deploying, 312, 314, 315–322
- object models for, 112
- parameter fields in, 125–130
- planning, 108–109, 112
- record selection for, 124–125
- viewing external report in, 114–115
- viewing report in, 117–121

Winforms.exe file, 19**Word (Microsoft)**

- exporting reports to, 136, 284
- printing reports from, 188

WSDL (Web Service Description Language), 195**X****X+2 button, 98–99****XLS files, exporting reports to, 136, 284****XML database files**

- as datasource, 33, 63
- errors with, 331

XML Report Web Services

- adding to application as a reference, 206–207
- binding to Report Viewer, 207
- consuming, 204–208
- copying to server, 208–209
- creating for multiple reports, 201–202
- creating for one report, 196–201
- creating Server Reports for, 202–204
- deployment of, 208–209
- errors with, 333–334
- features of, 5, 16, 194–196
- generic, included with Crystal Reports, 5
- naming, 200
- viewing report with generic Report Web Services, 202–204, 207–208

Web setup project for, 209

when to use, 196

Xor operator, 258**Xtreme Mountain Bike Company sample database, 21, 26****Y****YearFormat property, DateFieldFormat class, 302****YearToDate function, Basic Syntax, 262****Z****zoom events, 143, 190****Zoom method, Report Viewer, 180****ZoomEventArgs class, 114, 155****zooming report pages, 138–139, 180–181**