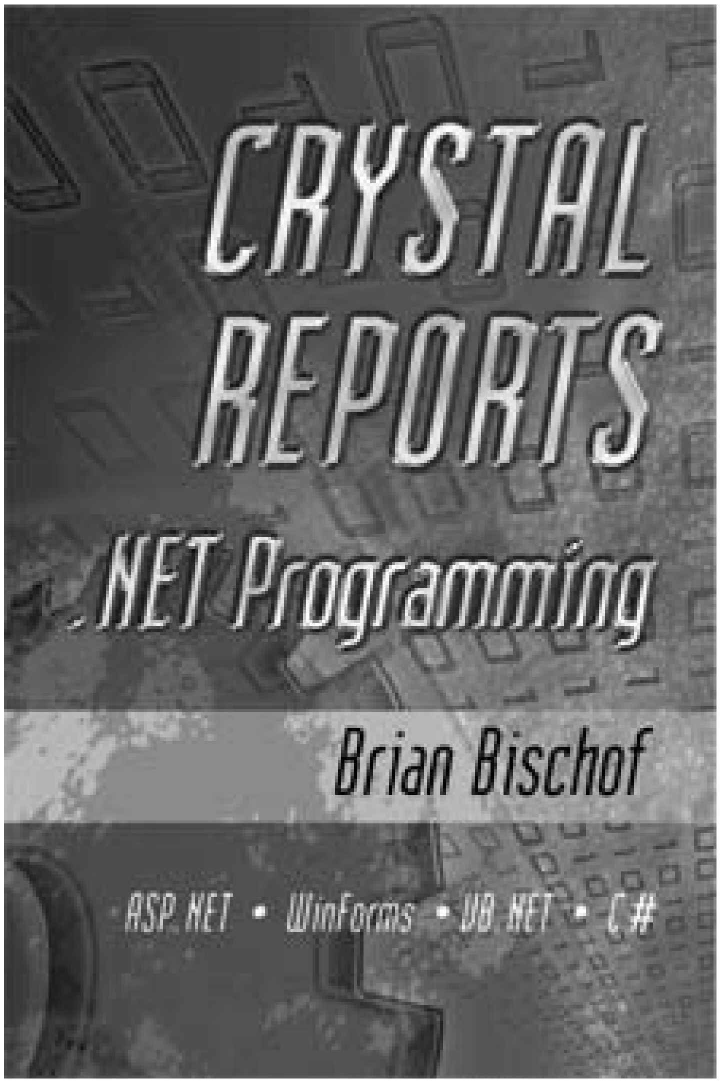
Excerpts from the book

Crystal Reports .NET Programming

By Brian Bischof
Copyright 2004



This free ebook gets you started learning Crystal Reports for .NET. The chapters in this book are direct excerpts from the book Crystal Reports .NET Programming (ISBN 0974953652). You get to learn a lot about Crystal Reports .NET and determine if the hardcopy book would benefit you. If you find this tutorial helpful, the book is for sale on www.Amazon.com. The complete table of contents is listed below. For more information see the book's website at www.CrystalReportsBook.com.

Crystal Reports .NET Programming
Table of Contents

PREFACE

With the release of Visual Studio .NET, Microsoft gave programmers the first powerful report writing tool that is completely integrated into the development environment. Crystal Reports has been included with previous versions of Visual Studio in the past, but never with this degree of integration. Programmers can now build a powerful reporting solution using the standard Visual Studio installation.

The one piece of the puzzle that is still missing is a solid set of documentation on how to use Crystal Reports .NET to create a sophisticated reporting solution. The help files are great, but they don't hold your hand and walk you through the various steps to be productive. Prior to this book being printed, all the Crystal Reports books available have focused their attention on the end user and treated programmers as they were an after-thought. Finding useful code is like looking for a needle in a haystack. This is the first and only book on the market written by a professional software developer for other software developers. Within these pages you will find countless code listings to make every code-junkie happy. Beginning programmers and advanced programmers alike will find that this book meets their needs.

Who this Book Is For

This book is for programmers that are new to Crystal Reports as well as programmers that are experienced with Crystal Reports. If you are new to Crystal Reports, you will be shown how easy it is to quickly create your first report and add it to your project. As you want to learn more reporting features you can turn to the appropriate chapter and see how easy it is to make your reports more professional. If you are experienced with Crystal Reports, you will be interested in learning what the .NET version of Crystal Reports lets you do as well as what its limitations are. Learning how to perform runtime report customization will let you take your reporting solution to the next level.

How the Book is Organized

This book is divided into two parts: Part I - Designing Reports and Part II - Programming Reports. Each part is designed for two different types of report development.

Part I - Designing Reports is for the user who has either never used Crystal Reports before or has used a previous version and wants to get up to speed on the .NET version. It walks you through the steps of creating reports using the report designer. You are also shown how to add sophistication to your reports by learning how to program with Crystal Syntax and Basic Syntax.

Part II - Programming Reports is for the advanced programmer who has mastered the art of designing reports and wants to take their reports to next level by customizing them during runtime. You will learn the intricacies of how the Crystal Reports object model is designed. This lets you take control of the report during runtime. Unlike Part I which is focused on using the Report Designer, Part II focuses on writing code with either VB.NET or C#.

Installing Crystal Reports

Crystal Reports for .NET is included with Visual Studio .NET. When you install Visual Studio .NET it is one of the tools installed by default. After installation, Crystal Reports is listed as one of the components that can be added to a project.

Crystal Reports for .NET is not included in the VB.NET or C# Standard versions. You need to purchase one of the Visual Studio .NET packages to get Crystal Reports.

Installing Service Packs

If you look at the Crystal Decisions support site for .NET, you'll see that there are dozens of known bugs with the product. As a result, Crystal Decisions is always making corrections and improvements to the product. It is critical that you go to their website on a regular basis to check for new downloads. In fact, you should put down the book right now and install the lastest service packs.

There are two types of downloads available: Service Packs and Hot Fixes. Service packs are released every six months. They include comprehensive bug fixes and additional features. Service packs have also been regression tested. Hot fixes are released at the beginning of each month. They include minor bug fixes and have not been as thoroughly tested as service packs.

Make sure you download the proper service pack for your version of Visual Studio. Service packs with a prefix of "CR10" are for Visual Studio 2002. Service packs with a prefix of "CR11" are for Visual Studio 2003.

Service Packs: http://support.CrystalDecisions.com/ServicePacks

Hot Fixes: http://support.CrystalDecisions.com/hot_fix_application_guide/

VB.NET and C# Code

The code examples in this ebook only show the VB.NET programming code. It is fairly easy to translate this code into C#. The hardcopy edition of the book has the complete C# code listed. If you are looking for a quick way to convert VB.NET to C# (or vice-versa), please see my book "The .NET Languages: A Quick Translation Guide" (ISBN 1-893115-48-8). It makes converting code easy.

Questions and Comments

Please email me your questions and comments to Comments@CrystalReportsBook.com. I can't respond to every email individually, but I will post new information on the book's website. Any comments about the book will be used to improve the 2nd edition that will come out for the next version of .NET.

PART I

Designing Reports

Learn how easy it is to design your first report and integrate it into a .NET application. After designing a couple sample reports for both Windows and ASP.NET, you begin to understand report layouts and adding new report objects. You'll quickly move beyond the report wizards and create reports using grouping and sorting, running totals, and subreports. If you want to perform dynamic formatting of report objects, learn how to program in Basic syntax or Crystal syntax. This chapter takes you from being a novice to intermediate report designer.

The code samples in Part I are deliberately kept simple so you can focus on learning how to design professional looking reports. When you're ready to tackle .NET runtime customization in VB.NET or C#, move to Part II of the book.

1 Introducing Crystal Reports

Visual Studio .NET is the first Windows development environment that gives developers a fully integrated and robust reporting solution. Crystal Reports is now installed with Visual Studio so developers can write applications that have reports seamlessly integrated into them. Starting with Visual Basic 3.0, Crystal Reports was included with the language, but not part of the default installation. It was also a stand-alone product that was independent of the programming language.

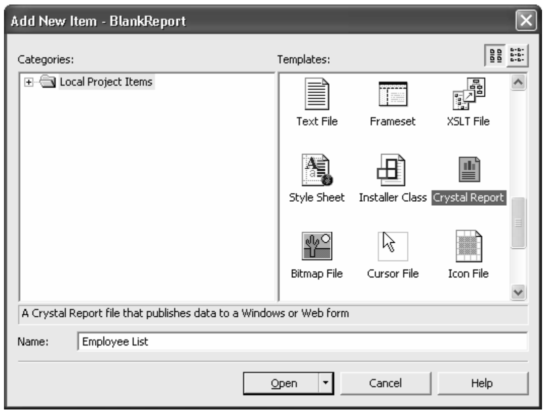
Over the years Microsoft has been including Crystal Reports with each version of Visual Basic. With version 6 they even wrote a Data Report component that was supposed to be a replacement for Crystal Reports. But it failed miserably.

With the release of Visual Studio.NET, Microsoft finally woke up to the needs of developers. They licensed Crystal Decisions to write a version of Crystal Reports to be the default reporting solution installed with .NET.[1] Built into the IDE, Windows developers now have the tools to write presentation-quality interactive reports.

Creating Your First Report

Before you learn about all the features of Crystal Reports, it is best to start by creating a quick report. Open Visual Studio and create a new project. This can be either VB.NET or C#, and it can be either a Windows Application or ASP.NET. Designing reports with Crystal Reports is independent of the application type. The following steps are the same for both types of applications.

Once the project is open, select Project | Add New Item. This displays the list of available templates. Scroll down near the bottom and select the Crystal Report template. Enter the name "Employee List". Figure 1-1 shows this dialog box for a Windows Application. Click Open to create the report.



A Crystal Report file that publishes data to a Windows or Web form

Name: Employee List

Figure 1-1. The Add New Item dialog box.

[1] Crystal Reports for .NET is a set of Runtime Callable Wrappers (RCW) around modified Crystal Reports 8.5 DLLs. They have been modified to support the Crystal Reports 9/10 file formats.

When the Crystal Report Gallery dialog box appears, accept the defaults of "Use Report Expert" and "Standard Report". This opens the Standard Report Expert shown in Figure 1-2.

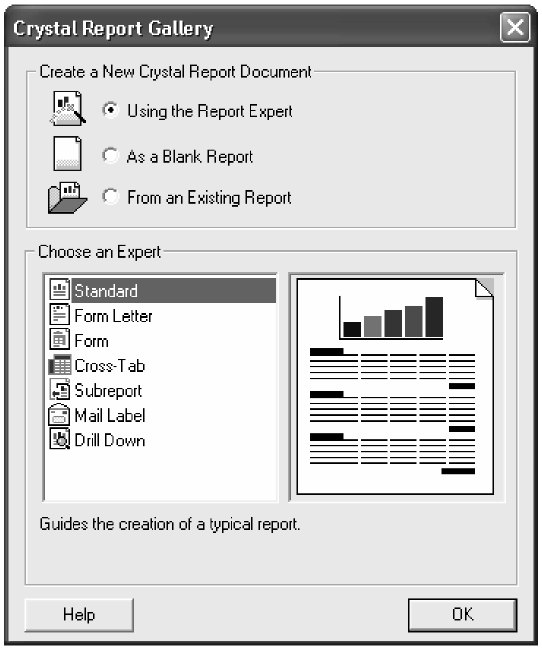


Figure 1-2. The Standard Report Expert dialog box.

The first tab shown is the Data tab. Select the Xtreme Sample Database (it comes with Crystal Reports) by clicking on the OLE DB (ADO) option in the Available Data Sources list. This brings up a new dialog box. Select Microsoft Jet 4.0 OLE DB Provider. On the next dialog box enter the database name (database name is the fully qualified file path). By default it is located at C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Database. Select Finish.

You are back at the Data tab of the expert. Click on the Tables node to expand and double-click on the Employee table name (or drag and drop it) to move it to the window titled Tables In Report. Click Next.

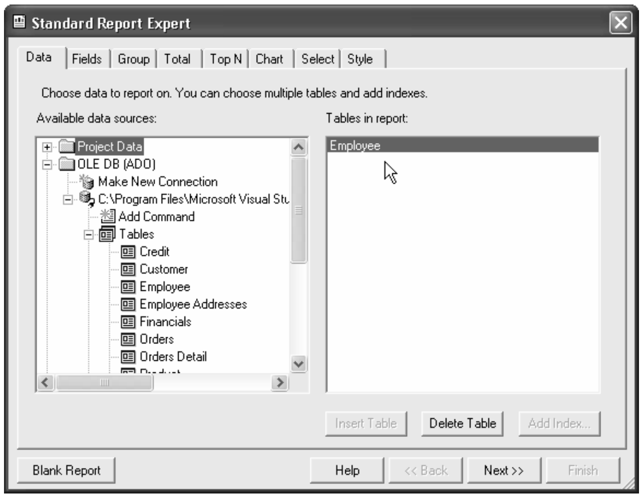
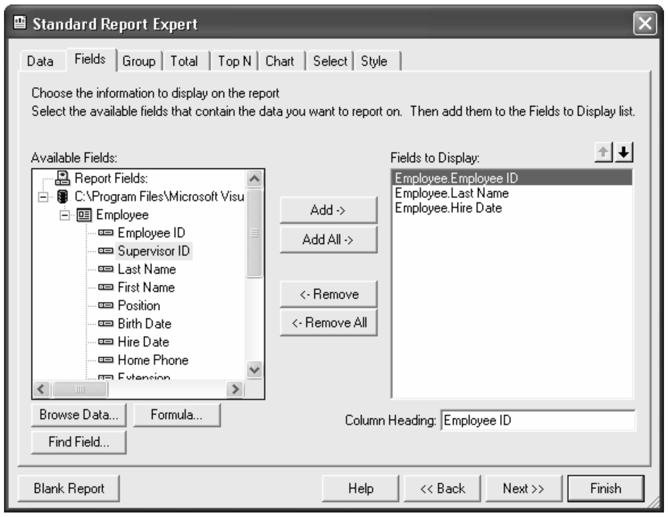


Figure 1-3. The Data tab of the report expert.

To display fields on the report, you have to select them at the Fields tab. Double click on the following fields to add them to the right window: Employee Id, Last Name, and Hire Date. This is shown in Figure 1-4.


Standard Report Expert

Choose the information to display on the report
Select the available fields that contain the data you want to report on. Then add them to the Fields to Display list.

Available Fields:
- Report Fields:
- C:\Program Files\Microsoft Visu
  - Employee
    - Employee ID
    - Supervisor ID
    - Last Name
    - First Name
    - Position
    - Birth Date
    - Hire Date
    - Home Phone
    - Extension

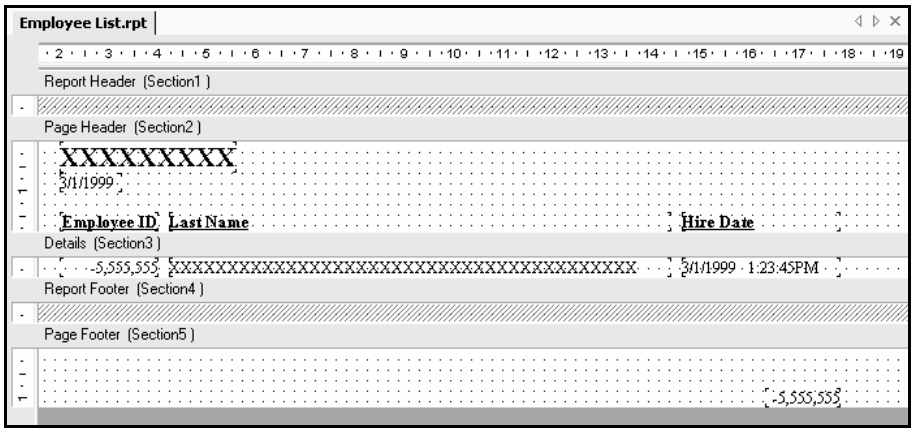
Fields to Display:
- Employee.Employee ID
- Employee.Last Name
- Employee.Hire Date

Column Heading: Employee ID

Figure 1-4. The Fields tab of the report expert.

At this point you could continue with the report expert to do things such as grouping and selecting which records to print. But for this simple example, ignore those tabs and click on the Style tab. Enter the title Employee List and keep the default style of Standard. Click Finish.

The report expert closes and builds a fully functioning report that is ready to run (Figure 1-5). If this report were to be used in a real application, the next step would be to modify the form so that it can preview and print the report.


Employee List.rpt

Report Header (Section1)

Page Header (Section2)
XXXXXXXXX
3/1/1999
Employee ID Last Name Hire Date

Details (Section3)
-5,555,555 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 3/1/1999 1:23:45PM

Report Footer (Section4)

Page Footer (Section5)
-5,555,555

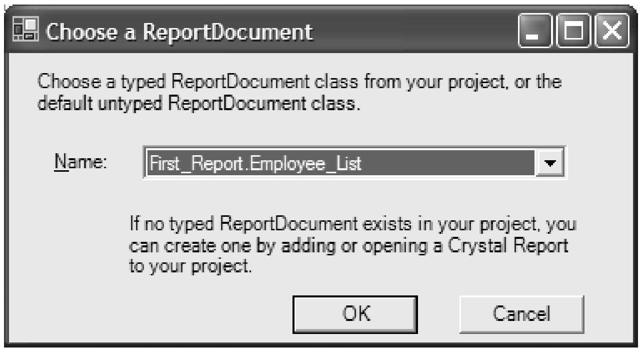
Figure 1-5. Employee List report file.

Previewing with a Windows Form

After creating the Employee List report in the previous section, you can preview it using either a Windows Form or and ASP.NET application. This section shows you how to preview it from a Windows Form. ASP.NET is covered in the next section.

Previewing the report requires modifying the form. When you created the new project, Form1 should have been automatically added to the project for you. Open Form1 in design mode and add a CrystalReportViewer control to it. This is normally listed as the last component in the Windows Forms section of the Toolbox. Resize the viewer so that it fills up the entire form. Do this by finding its Dock property and clicking on the drop-down box. Click on the middle square so that the property is set to Fill.

The viewer control has many ways to preview reports.[2] This example uses the ReportDocument component because it is the easiest of the choices. Add the ReportDocument component to the form by double-clicking on it from the Componets section of the Toolbox. A Choose a ReportDocument dialog box automatically appears, allowing you to select which report to display. The dropdown control lists all the reports that are part of your project. For this example, it only shows the Employee List report.



Choose a ReportDocument

Choose a typed ReportDocument class from your project, or the default untyped ReportDocument class.

Name: First_Report.Employee_List

If no typed ReportDocument exists in your project, you can create one by adding or opening a Crystal Report to your project.

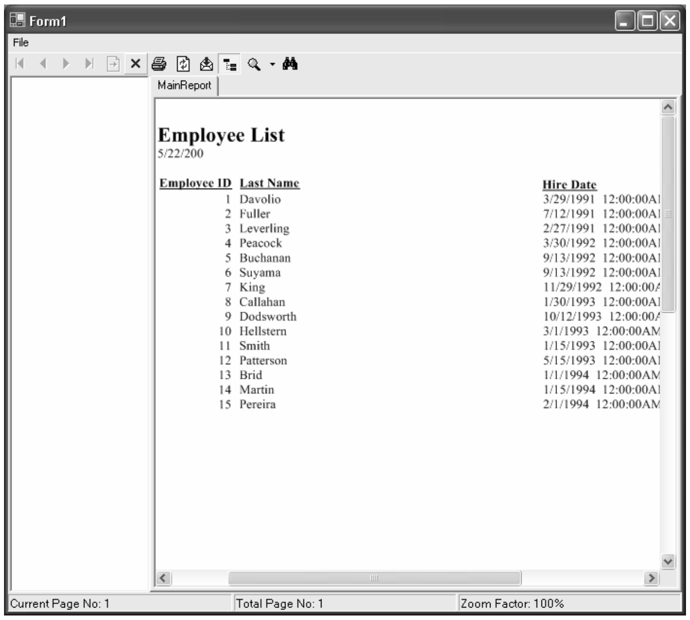
Figure 1-6. The ReportDocument dialog box.

Select the Employee List report and click the OK button. This adds the ReportDocument component to your form.

You have to tell the viewer that the report it is going to preview is in the ReportDocument component. Look at the Properties Window and find the ReportSource property. It has a dropdown control which lists the ReportDocument components on the form. In this example, the Employee List will be the only one listed. Click on the Employee_List component to select it.

Run the application. When the form loads it automatically instantiates the Employee List report object and display it using the viewer. Print it by clicking on the printer button in the menu bar. The preview of the Employee List report is shown in Figure 1-6.

[2] Chapter 3 gives a complete explanation of using the viewer control.



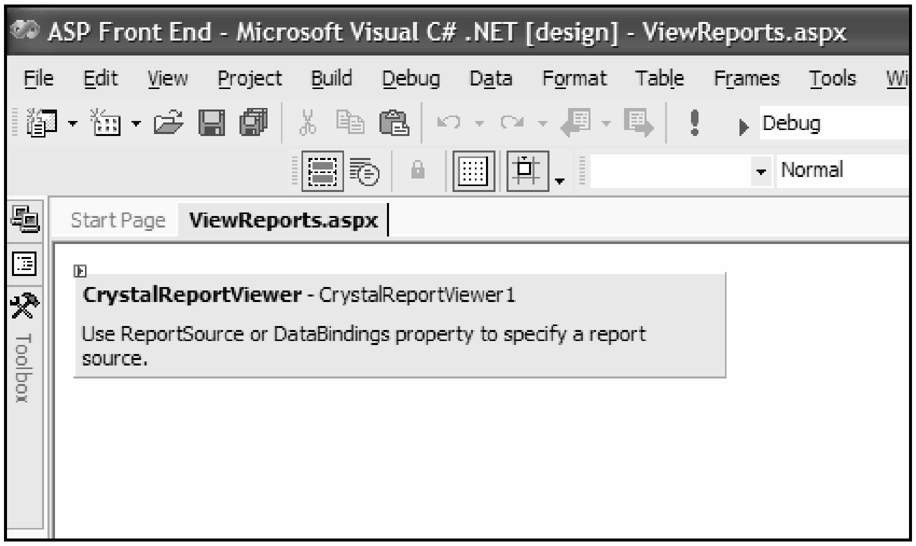
```
Employee List
5/22/200

Employee ID  Last Name                                   Hire Date
          1  Davolio                                     3/29/1991  12:00:00AI
          2  Fuller                                      7/12/1991  12:00:00AI
          3  Leverling                                   2/27/1991  12:00:00AI
          4  Peacock                                     3/30/1992  12:00:00AI
          5  Buchanan                                    9/13/1992  12:00:00AI
          6  Suyama                                      9/13/1992  12:00:00AI
          7  King                                        11/29/1992  12:00:00A
          8  Callahan                                    1/30/1993  12:00:00AI
          9  Dodsworth                                   10/12/1993  12:00:00A
         10  Hellstern                                   3/1/1993  12:00:00AM
         11  Smith                                       1/15/1993  12:00:00AI
         12  Patterson                                   5/15/1993  12:00:00AI
         13  Brid                                        1/1/1994  12:00:00AM
         14  Martin                                      1/15/1994  12:00:00AI
         15  Pereira                                     2/1/1994  12:00:00AM
```

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

Figure 1-6. Employee List report output in preview mode.

You can see that this quick report isn't perfect. The Hire Date column shows times as always being 12:00:00 AM and the Group Tree is visible even though there aren't any groups in the report. As is usually the case, the report expert gives you a good basis for writing a report, but you still need to make some changes to clean it up.

Previewing a report with ASP.NET requires adding a CrystalReportViewer control to the web page and setting its data bindings. Open the default web page in design mode and add a viewer control to it. The viewer is located near the end of the Toolbox under the Web Forms tab. Once you add it to the form you will see a medium sized rectangle on the web page.



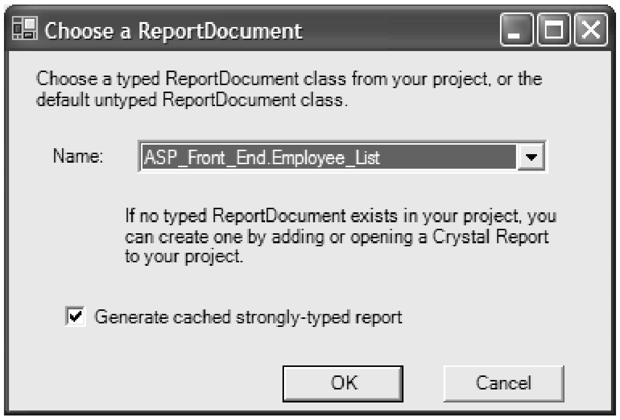
CrystalReportViewer - CrystalReportViewer1

Use ReportSource or DataBindings property to specify a report source.

Figure 1-7. The ASP.NET report viewer control.

Notice that the web viewer control looks totally different than the Windows viewer control. The web viewer is simply a placeholder for where the report will be displayed.

Add a ReportDocument component to the web page by double-clicking on the ReportDocument component. The ReportDocument component is listed in the Components section of the Toolbox. As mention in the previous section, when you add a ReportDocument component to your form, it automatically displays the Choose a ReportDocument dialog box. This lets you select which report will be displayed. The dropdown control lists all the reports that are part of your project. For this example, it will only show the Employee List report.


Choose a ReportDocument

Choose a typed ReportDocument class from your project, or the default untyped ReportDocument class.

Name: ASP_Front_End.Employee_List

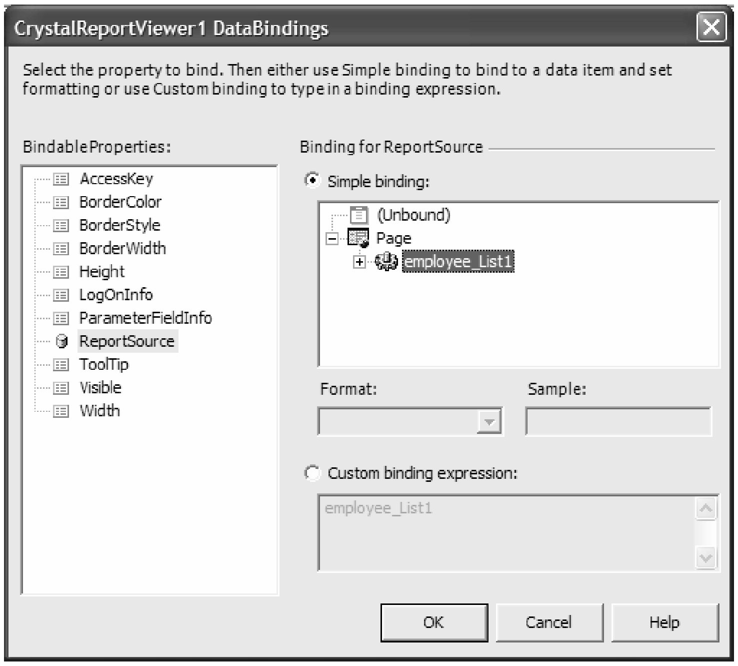
If no typed ReportDocument exists in your project, you can create one by adding or opening a Crystal Report to your project.

☑ Generate cached strongly-typed report

Figure 1-8. The ReportDocument dialog box.

Select the Employee List report and click the OK button. This adds the ReportDocument component to your form.

You have to associate the ReportDocument component with the viewer control so that the viewer knows which report to display. Look at the viewer's properties and at the very top is the DataBindings property. Click on it and then click on the ellipses button to open the DataBindings dialog box.


CrystalReportViewer1 DataBindings

Select the property to bind. Then either use Simple binding to bind to a data item and set formatting or use Custom binding to type in a binding expression.

Bindable Properties:
- AccessKey
- BorderColor
- BorderStyle
- BorderWidth
- Height
- LogOnInfo
- ParameterFieldInfo
- ReportSource
- ToolTip
- Visible
- Width

Binding for ReportSource

(●) Simple binding:
- (Unbound)
- Page
  - employee_List1

Format:
Sample:

( ) Custom binding expression:
employee_List1

Click on the ReportSource property listed on the left. Expand the Page item in the Simple Binding window. It shows you the reports that were added to the page using the ReportDocument components. In this example, only the Employee List report is shown.

Note

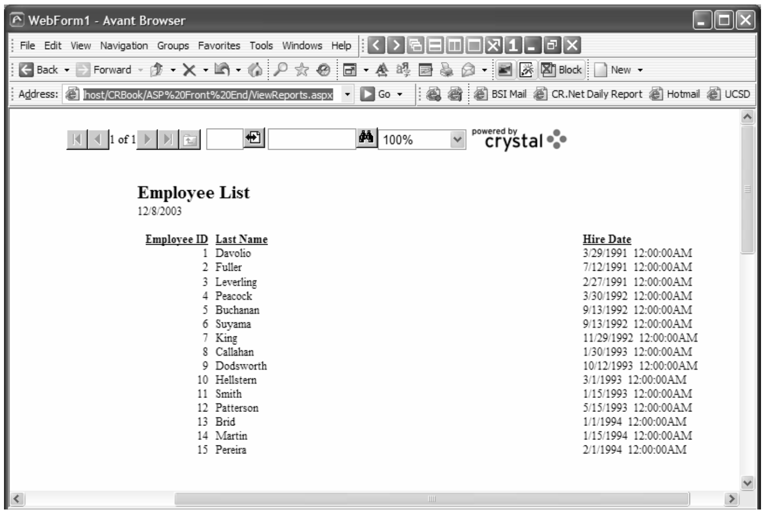
Don't skip the above step. I keep getting emails from people asking why their report doesn't print and I find out that they skipped this step. Make sure the ReportSource property is selected before going further.

Click on the Employee List report item and click the OK button. A preview of the report is immediately displayed. Unlike Windows development, the viewer control shows you what the report looks like while you are in design mode. This a great feature for web developers!

At this point, you've performed all the same steps that were shown during the previous example for previewing with a Windows application. However, if you ran your ASP.NET application now, the report wouldn't be shown. To make the report display on the web page you have to call its DataBind() method. Do this in the Page_Load() event.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    CrystalReportViewer1.DataBind()
End Sub
```

After typing in this code, run the web application and you'll see the Employee List report automatically previewed when the page opens.


WebForm1 - Avant Browser
File Edit View Navigation Groups Favorites Tools Windows Help
Back Forward Block New
Address: host/CRBook/ASP%20Front%20End/ViewReports.aspx Go BSI Mail CR.Net Daily Report Hotmail UCSD
1 of 1 100% powered by crystal

Employee List
12/8/2003

Employee ID	Last Name	Hire Date
1	Davolio	3/29/1991 12:00:00AM
2	Fuller	7/12/1991 12:00:00AM
3	Leverling	2/27/1991 12:00:00AM
4	Peacock	3/30/1992 12:00:00AM
5	Buchanan	9/13/1992 12:00:00AM
6	Suyama	9/13/1992 12:00:00AM
7	King	11/29/1992 12:00:00AM
8	Callahan	1/30/1993 12:00:00AM
9	Dodsworth	10/12/1993 12:00:00AM
10	Hellstern	3/1/1993 12:00:00AM
11	Smith	1/15/1993 12:00:00AM
12	Patterson	5/15/1993 12:00:00AM
13	Brid	1/1/1994 12:00:00AM
14	Martin	1/15/1994 12:00:00AM
15	Pereira	2/1/1994 12:00:00AM

Figure 1-9. The Employee List report shown in an ASP.NET application.

Examining the Report Designer

Each report within your application is just like any other component that your application uses. It is listed in the Solution Explorer window as a class in your project. When you double-click it, it opens the report in design mode and you can make changes to it.

Each report starts with five sections: Report Header, Page Header, Detail, Page Footer, and Report Footer. These sections are described in Table 1-1.

Table 1-1. Report sections.

Section	Description
Report Header	Appears at the top of the first page of the report.
Page Header	Appears after the Report Header on the first page. On all the remaining pages it appears at the top of the page.
Group Header	Appears at the beginning of each new group.
Details	The row that displays the record information. There is usually one detail row for every record in the table.
Group Footer	Appears after all records of a group have been printed.
Page Footer	Appears at the bottom of each page.
Report Footer	Appears as the bottom of the page for the last page in the report.

To the left of the report layout is the Toolbox as shown in Figure 1-10. When you have the report designer open, there are only a few controls available in the Toolbox. They are the Text Object, Line Object, and Box Object. These are the most basic of the controls available.

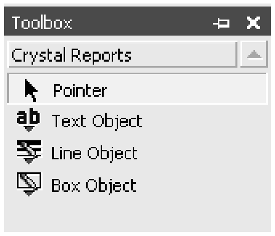


Figure 1-10. Crystal Report's toolbox.

Oddly enough, there are more controls that can be used on a report, but they aren't listed in the toolbox. You have to right-click on the report and select the Insert menu option. This menu is shown in Figure 1-11. There are ten controls on it that can be added to a report as well as a lot of special fields that display report specific information (e.g. page number, print date, etc.). These controls are described in Chapter 2.

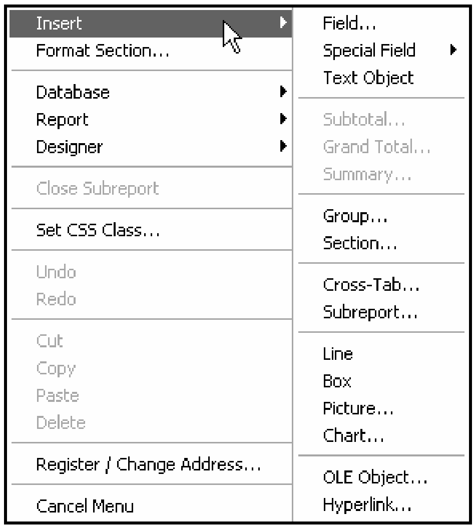


Figure 1-11. Crystal Report's Insert menu.

In the bottom right hand corner of the IDE is the Properties window. As expected, these properties are only applicable for the control that has the focus.

The CrystalReportViewer Control

The CrystalReportViewer control is used on a form to display a report. You have to use this control when you want to preview and print a report. It is found in the form's Toolbox as the last control listed. To access it, you have to use the down arrow to scroll down to it.

The CrystalReportViewer control is fully customizable. Since it is the only way to display a report in your application, you may need to customize it for some applications. You don't want a user to feel like they are looking at a third-party control when using your application. Each of the buttons can turned on or off in design mode or during runtime. You can also add your own buttons to a form and have them manipulate the report layout and respond to user events.

Two-Pass Report Processing

Crystal Reports processes reports in three stages. This is called the Two-Pass Report Processing Model. The first pass creates the primary data to be printed. During the second pass this data is further processed to finalize it for printing.

The first pass reads individual records one at a time and calculates all formulas. This pass only calculates the formulas that are based on raw data within a record or that perform simple calculations. As each record is read and the formulas are calculated, the results are stored in a temporary file to be used during the second pass.

After the first pass is finished, Crystal Reports performs a second pass where it evaluates all summary functions on the data. This wasn't possible during the first pass because all the data had not been read yet. During the second pass all the raw data has already been read into the temporary file and it can be evaluated and summarized as a whole.

After the second pass is finished, the report engine calculates the total number of pages if the report needs it or if the special field Page N of M is used.

Understanding the type of data that is processed during the Two-Pass Processing Model is essential to being able to write formulas and summarize data. Different chapters throughout this book reference this model for explaining when you can and can’t use certain reporting functionality.

PART II
Programming Reports

Advanced report designers aren't satisfied with using the built-in functions to customize reports. They want to integrate their .NET applications with Crystal Reports during runtime. Part II shows you how to seamlessly integrate your reports into .NET with runtime customization of report objects and modifying parameters. Dynamic data connections let you connect to virtually any data source. For the most advanced reporting functionality, develop Report Web Services or upgrade to the stand-alone version of Crystal Reports and use .NET to program the RDC and the RAS. Part II shows you how to take your reporting skills to the expert level.

14

Learning the Report Object Models

Part I of this book taught the details of designing reports by laying out the report objects in the different report sections and connecting to data sources. When the report was finished its design and layout stayed the same every time it was run. The data that it prints will change, but the report format is locked. With Crystal Reports .NET, reports have the flexibility to be dynamic. .NET programmers have full access to the properties of a report and the underlying report objects. These properties can be read and many of them can be written to. You get the power to create a reporting solution that takes user input and customizes each report prior to printing it. This can range from changing the formatting of report objects, modifying the grouping and sorting, and changing the data source. The more you learn about runtime customization the more you will find out what you can do. This chapter serves as the foundation for building your knowledge throughout the rest of the book.

The reason that you have so much power to modify reports is because .NET treats every report as an object-oriented class. The entire object model is exposed to your .NET program. Whether you program with VB.NET or C# isn't important. The object model can be accessed by any of the .NET languages.

There are three ways to use the Crystal Reports object model. The first is to use the ReportDocument class to reference virtually every class and property of the report. The second way is to use the methods and properties of the CrystalReportViewer control. When compared to the ReportDocument class, the viewer only has a small subset of properties and methods. The viewer lets you modify the properties that effect logging in to a data source, setting report parameters, and deciding which report to preview. The last way to work with the object model is to subscribe to the events that are triggered while the report is previewed and printed. These events are useful for knowing what part of the report is being looked at and what the user is doing. This chapter goes into detail on all three ways of working with the report classes.

If you have experience with Crystal Reports 8.5 Developer Edition, then you are probably familiar with modifying reports during runtime. One of the key features of the Developer Edition was that the Visual Studio 6.0 developer had a greater amount of control over the report during runtime. In fact, a developer could write an entire report from scratch during runtime! The report classes that come with .NET are not this sophisticated. While you do have a lot of flexibility with modifying an existing report, you are limited to using the existing report objects. You can't create new report objects nor can you change the fields that the current objects link to. If you want to create reports and add new report objects, you have to purchase a separate developer's license. However, this license is extremely expensive and is only practical for companies with a large budget.

Basic Customization

No matter what type of runtime customization you want to perform, there are three basic steps that you need to know. Customizing reports always starts with the same premise: declare and instantiate a ReportDocument object, modify its properties, and print or preview it.

Chapter 3 gave a thorough explanation of the different ways to integrate reports into an application (Strongly-Type versus Untyped). If you need a refresher, refer to the sections on binding reports for a discussion of the pros and cons of the different methods of binding reports.

Note

The code that implements runtime customization is the same for both WinForms development and ASP.NET applications. The ReportDocument class is used with both types of applications. ASP.NET will only be mentioned for special circumstances.

Step 1: Declare and instantiate the ReportDocument object variable.

An object variable can either instatiate the report class directly (Strongly-Typed reports) or it can instantiate the ReportDocument class and then load the report into memory (Untyped reports). Since runtime customization requires the use of Strongly-Typed reports, the examples in this chapter will use this binding method.

Note

In the examples throughout this chapter and the rest of the book, the report class being referenced is called CrystalReport1. This is the default report name that is given by the report wizard. When you implement the sample code in your applications, replace the name CrystalReport1 with the class name of the report you want to print. All the code samples are generic and will work with every report.

```
'Declare and instantiate an object variable of the report class
Dim MyReport As New CrystalReport1
```

Step 2: Modify the properties of the report object.

After instantiating the report variable, all the properties and methods of the ReportDocument object are available to you. Set the properties that need to be modified. The different properties of the report object are explained throughout all the chapters in Part II of this book.

In this example, the record selection formula is changed. The report variable MyReport (from Step 1) is used to get a reference to the DataDefinition object and change its RecordSelectionFormula property.

```
MyReport.DataDefinition.RecordSelectionFormula = "{Orders.Order Date}>#01/01/2004#"
```

The above line of code is very simplistic for purposes of illustrating how to change a property of the ReportDocument class. Real applications aren't so simple and you can save yourself effort by using the same code in different projects. A lot of times you will find yourself taking a specific piece of code and rewriting it so that it can be generic enough to be used by multiple applications or put into a reporting library. In many of the examples in this book, I do this by writing a method that gets passed the report object and any necessary parameters. The method modifies the properties of the report object that was passed to it and exits. In these examples, the method's code will be shown but not the code that calls it. I won't repeat the code that declares and initializes the report variable and previews it. It is assumed that you know that these methods should be called after the report object is declared and initialized, but before previewing the report. If you need a refresher, you can refer back to this chapter.

Step 3: Print or preview the report.

To preview the report, pass the report variable to the viewer control. To print the report, call the PrintToPrinter() method of the report variable. To be consistent, my examples always preview the report. Thus, the sample code will assign the report variable to the viewer control.

```
CrystalReportViewer1.ReportSource = MyReport
```

Those three steps are always used for performing runtime customization. The next question you might be asking yourself is where to put the code. You can put this code anywhere in the form. If you want to load the form immediately and customize the report, put it in the Load() event. There are time that you are going to call this code after the user has entered various data that specifies how the report should be customized. If that's the case, put the code in response to the click event of an OK button that confirms they are finished inputting data. The following code sample shows you a complete code listing and it demonstrates where to put the code for calling a generic method to modify report properties.

Listing 14-1. A template for modifying reports.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
  Dim MyReport As New CrystalReport1
  'Call all report modification code here.
  'As mentioned in Step 2 above, this can be a method that is passed the report variable.
  'For illustration purposes, I'm calling a generic method that changes
  'the report title. The code for ModifyTitle() isn't shown here.
  ModifyTitle(MyReport, "Runtime Demo")
  CrystalReportViewer1.ReportSource = MyReport
End Sub
```

If you are writing an ASP.NET application, this code can be put in the Page_Load() event. See Chapter 3 for a more information about writing ASP.NET applications.

Note

All report customization must be done prior to previewing or printing the report. No changes are allowed once the report is generated. For example, you can't use .NET to change the way a field is formatted depending upon the current group value. Making dynamic report changes while the report is running requires writing formulas and using conditional formatting (discussed in Chapters 7, 8, and 9).

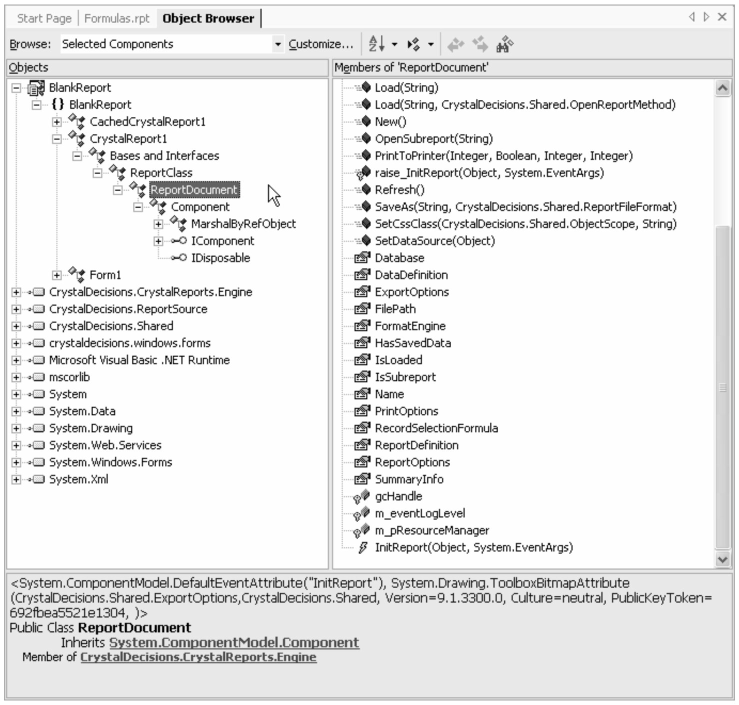
ASP.NET Template

Printing reports within an ASP.NET application requires a different template than the Windows template shown in Listing 14-1. To see the template for ASP.NET, please refer to the hardcopy version of Crystal Reports .NET Programming.

The ReportDocument Object

The ReportDocument class is the base class for all reports. Its properties give an application the ability to thoroughly examine all the report objects. Many of these properties, but not all of them, have write capabilities so that you can modify their values.

Each report is a class that inherits from the ReportDocument class. Figure 14-1 shows the Object Browser window with the class for a blank report, CrystalReport1. You can see that ReportClass is the base class for the report. This class is derived from the ReportDocument class. The members listed to the right of the figure belong to the ReportDocument class.



```
BlankReport
  {} BlankReport
    CachedCrystalReport1
    CrystalReport1
      Bases and Interfaces
        ReportClass
          ReportDocument
            Component
              MarshalByRefObject
              IComponent
              IDisposable
    Form1
CrystalDecisions.CrystalReports.Engine
CrystalDecisions.ReportSource
CrystalDecisions.Shared
crystaldecisions.windows.forms
Microsoft Visual Basic .NET Runtime
mscorlib
System
System.Data
System.Drawing
System.Web.Services
System.Windows.Forms
System.Xml
```

```
Load(String)
Load(String, CrystalDecisions.Shared.OpenReportMethod)
New()
OpenSubreport(String)
PrintToPrinter(Integer, Boolean, Integer, Integer)
raise_InitReport(Object, System.EventArgs)
Refresh()
SaveAs(String, CrystalDecisions.Shared.ReportFileFormat)
SetCssClass(CrystalDecisions.Shared.ObjectScope, String)
SetDataSource(Object)
Database
DataDefinition
ExportOptions
FilePath
FormatEngine
HasSavedData
IsLoaded
IsSubreport
Name
PrintOptions
RecordSelectionFormula
ReportDefinition
ReportOptions
SummaryInfo
gcHandle
m_eventLogLevel
m_pResourceManager
InitReport(Object, System.EventArgs)
```

```
<System.ComponentModel.DefaultEventAttribute("InitReport"), System.Drawing.ToolboxBitmapAttribute
(CrystalDecisions.Shared.ExportOptions,CrystalDecisions.Shared, Version=9.1.3300.0, Culture=neutral, PublicKeyToken=
692fbea5521e1304, )>
Public Class ReportDocument
        Inherits System.ComponentModel.Component
  Member of CrystalDecisions.CrystalReports.Engine
```

Figure 14-1. Object Browser view of the ReportDocument class.

The ReportDocument class has seven other classes that it references. Figure 14-2 shows the ReportDocument object model. The class thoroughly exposes all the objects of a report. Since the coverage is so broad and hits many topics covered in this book, the relevant classes are covered in different chapters. This chapter gives you an overview of all the classes and goes into detail on the two generic classes SummaryInfo class and ReportOptions class.

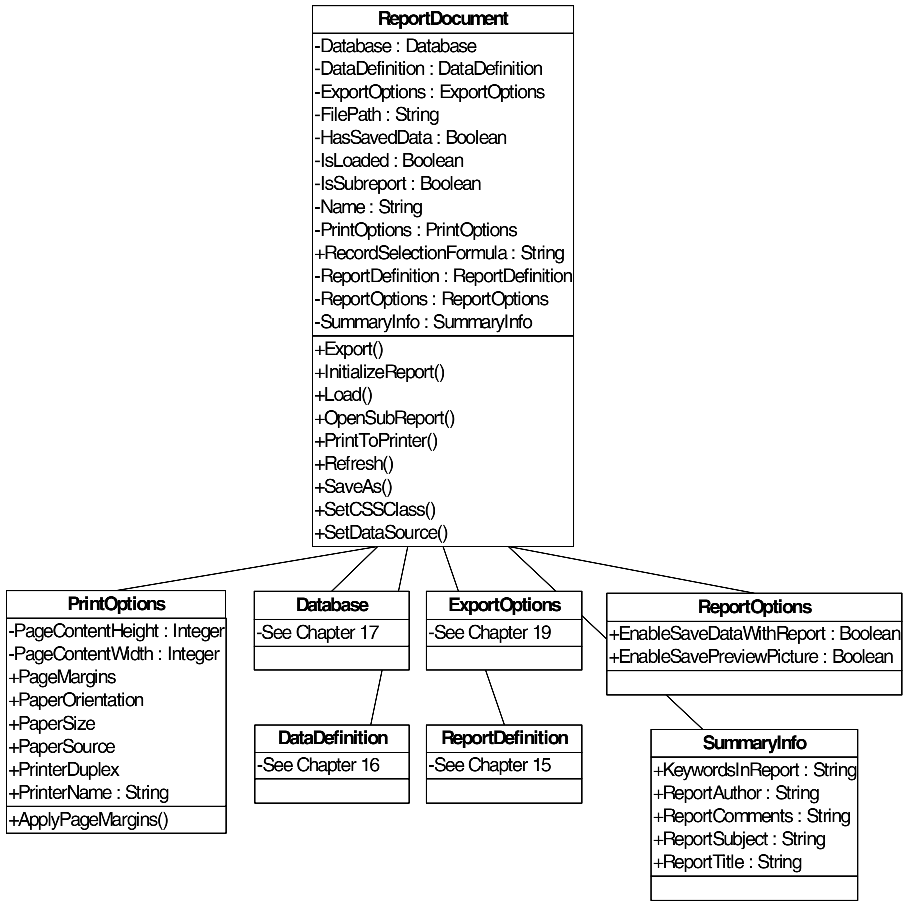


Figure 14-2. The ReportDocument object model.

Every report in Visual Studio is saved as a class. Since it is a class, you have to declare an object variable and instantiate it. There are two ways to create this object variable. You can declare and instantiate it yourself, or you can add a ReportDocument component to your form. Both of these methods were discussed in Chapter 3. Either method gives you access to the properties of the object variable to manage the different collections.

Retrieving Summary Information

Every report has a variety of summary information saved with it. This information is set by the report designer and it usually consists of things such as the report author, the report title and report comments. This information is set during design mode by right-clicking on the report and selecting Report | Summary Info. The class that maintains this summary information is the SummaryInfo class. Although you can override the property values, more than likely, you will only want to read from these values. This information was set by the report designer and won't need to change during runtime. There are only a half dozen properties that this class exposes (see Table 14-1).

Table 14-1. Properties of the SummaryInfo class.

Property	Description
KeywordsInReport	The keywords in the report.
ReportAuthor	The author of the report.
ReportComments	Comments about the report.

ReportSubject	The subject of the report.
ReportTitle	The title of the report.

The following example displays a message box showing the author of a report. You can see that accessing these properties is very simple. CrystalReport1 is the class for a generic report. Replace it with the class name of your own report.

```
Dim MyReport As New CrystalReport1
MessageBox.Show(MyReport.SummaryInfo.ReportAuthor)
```

Setting the Report Options

The ReportOptions class only has a few properties that can be set. They are listed in Table 14-2. This information is set during design mode by right-clicking on the report and selecting Designer | Default Settings and selecting the Reporting tab. These properties were discussed in Chapter 2.

Table 14-2. Properties of the ReportOptions class.

Property	Description
EnableSaveDataWithReport	Saves the latest data with the report. Allows report to be opened without a live data connection.
EnableSavePreviewPicture	Saves a thumbnail picture of the report.
EnableSaveSummariesWithReport	Saves data summaries with the report.

The following example sets the EnableSaveDataWithReport property to True.

```
Dim MyReport As New CrystalReport1
MyReport.ReportOptions.EnableSaveDataWithReport = True
```

Connecting to the Data Sources

The Database class is used for examining the tables used in a report and their relationships with each other. It's also used for setting the login information before opening the report. This class is described in Chapter 17.

Modifying the Printing Options

The PrintOptions class stores the options for how a report is sent to the printer. This can consist of the destination printer, the paper orientation or the page margins. This is normally set during design mode. While the majority of an application's reports will use the same settings, you can override the default settings for specific reports. Table 14-3 lists the properties.

Table 14-3. PrintOptions properties.

Property	Description
PageMargins	Gets the page margins.
ApplyPageMargins()	Sets new page margins.
PaperOrientation	Switch between Landscape and Portrait.
PaperSize	Set the paper size using pre-defined size constants.
PaperSource	Set the tray that the paper is printed from.

PrinterName	Change the printer by passing a string that exactly matches the printer name listed in the Printers Control Panel.

Each of these properties is easy to modify. In same cases you will have to use a predefined constant to set the property (e.g. PaperOrientation and PaperSize).

Caution

Changing the printer name can cause the report output to be scrambled. Each printer uses a unique printer language for producing output. If the new printer doesn't use the same printer language as the default printer that the report was designed to use, then the report will not print correctly. For example, if a report was designed for use with an HP printer, then it is okay to switch between similar models of an HP printer. But printing this report to Acrobat PDFWriter will result in an unreadable PDF file.

Listing 14-4. Change a report's printer settings.

```
Dim MyReport As New CrystalReport1
MyReport.PrintOptions.PaperOrientation = CrystalDecisions.[Shared].PaperOrientation.Landscape
MyReport.PrintOptions.PrinterName = "HP LaserJet510"
MyReport.PrintToPrinter(1, False, 0, 0)
```

Exporting Reports

If your application only sends reports to the printer, it is lacking the functionality to make your data accessible to a variety of applications. Crystal Reports lets you export a report in many different formats so that different applications can read the data. For example, you can export report data to an Excel spreadsheet so that an end user can perform a statistical analysis on the data. The ExportOptions class has the properties that specify the exporting options. This is discussed in Chapter 19.

Referencing and Formatting the Report Objects

The ReportDefinition class is responsible for maintaining the collections of the basic report objects. These objects consist of the Areas collection, Sections collection and the ReportObjects class. Each Section class contains the report objects that are within that section. You can modify the formatting properties of any of these objects. This is discussed in Chapter 15.

Changing Report Objects

Every report can have many types of fields that are used to generate the report, but don't have to appear directly on the report. Some examples are grouping fields, parameter fields, and formula fields. Even though these fields may not be shown directly on the report, they are updateable during runtime and can be used to change the report's appearance. For example, you can change the grouping field so that the report sorts and summarizes in a new way. The DataDefinition class manages the collections that control these aspects of the report. These collections are discussed in the appropriate chapters throughout the book.

CrystalReportViewer Object Model

Previewing reports is done with the CrystalReportViewer control. The viewer can be used as an alternative to the ReportDocument class for modifying reports during runtime. It is a lightweight control and only exposes a few properties. You can use it when you only need want to perform basic tasks.

To see the CrystalReportViewer object model and how to program it, please refer to the hardcopy version of Crystal Reports .NET Programming.

Responding to Events

Report classes are built with events that let you respond to the actions that the user is doing as they preview a report. Your application can subscribe to these events and be alerted when they occur. The events that can be subscribed to are primarily associated with the actions that a user takes while previewing a report. Since a user can only preview a report using the CrystalReportViewer, the events are written for this class. The ReportDocument class only has the InitReport() event that can be subscribed to. Table 14-4 lists the reporting related events.

Note

The CrystalReportViewer also has the standard events associated with all controls (e.g. Click, GotFocus, etc.) However, these are not unique to Crystal Reports, so if you need more information about them, please consult MSDN.

Table 14-4. The primary events for reports.

Event	Description
InitReport()	Fired after a report has been successfully loaded. This is the only event available for the ReportDocument class. This is not available for the CrystalReportViewer class.
Drill()	Fired when the user drills down on a field.
DrillDownSubReport()	Fired when the user drills down on a subreport.
HandleException()	Fired when an exception occurs.
Navigate()	Fired when a user moves to another page on the report.
ReportRefresh()	Fired when the user refreshes the report data.
Search()	Fired when the user enters a search string.
ViewZoom()	Fired when the user changes the zoom percentage.

The Drill() event is fired whenever a user clicks on a field to drill down on it. It passes an object of type DrillEventArgs. This object can be examined to find the group level the user is currently looking at as well as the new group level being moved to. Table 14-5 lists the properties of the DrillEventArgs event type.

Table 14-5. Properties of the DrillEventArgs event type.

Property	Description
CurrentGroupLevel	Returns an integer representing the current group level.
CurrentGroupName	Returns a string representing the name of the current group level.
NewGroupLevel	Returns an integer representing the new group level.
NewGroupName	Returns a string representing the group level name.

The DrillDownSubReport() event is similar to the Drill() event and it is fired when the user drills down on a subreport. Although the functionality is similar, this event passes an object of the

DrillDownSubreportEventArgs type. It gives you information such as the subreport name and the page number. Table 14-6 lists the properties of this event type.

Table 14-6. Properties of the DrillDownSubreportEventArgs event type.

Property	Description
CurrentSubreportName	The name of the current subreport.
CurrentSubreportPageNumber	The page number that the subreport is on.
CurrentSubreportPosition	Returns a Point object that tells the position of the subreport on the viewer.
Handled	Set to true if you do not want the subreport to be drilled down to.
NewSubreportName	The name of the new subreport.
NewSubreportPageNumber	Sets the page number to drill down into.
NewSubreportPosition	Returns a Point object that tells the position of the new subreport on the viewer.

The HandleException() event is used for capturing exceptions and handling them. This is discussed in detail in the section Handling Exceptions.

The Navigate() event is fired when the user moves to another page in the report. This can be done by paging forward through the report or jumping to the beginning or end of the report. Table 14-7 lists the properties for the NavigateEventArgs event type.

Table 14-7. Properties of the NavigateEventArgs event type.

Property	Description
CurrentPageNumber	The page number that the report is on.
Handled	Set to True if you do not want to move to the new page.
NewPageNumber	The page number that the user is moving to.

The ReportRefresh() event is fired when the user refreshes the report data. The only property for this event is the Handled property. It is the same as the other events.

The Search() event is fired when the user searches for text within the report. Table 14-8 lists the properties for this event type.

Table 14-8. Properties of the SearchEventArgs event type.

Property	Description
Direction	Gets or sets the direction to be backward or forward. Use a variable of the SearchDirection type.
Handled	Set to True if you do not want to search for the text.
PageNumberToBeginSearch	Gets or sets the page number to start searching.
TextToSearch	Gets or sets the string to search for.

The ViewZoom() event is fired when the user changes the zoom level of the preview image. This event lets you find out the current zoom level and what the new zoom level will be. Table 14-9 lists the properties for this event type.

Table 14-9. Properties of the ZoomEventArgs event type.

Property	Description
CurrentZoomFactor	Gets the current zoom factor.
Handled	Set to true if you do not want to change the zoom factor.
NewZoomFactor	Gets the new zoom factor.

Handling Exceptions

Crystal Reports gives you the ability to handle any errors that might occur while printing and previewing. The benefit to handling the error yourself is that you can customize the error handling process. For example, you could write the error to a log file or you can gracefully exit the process without throwing an error message at the user.

The CrystalReportViewer class has a HandleException() event that you can subscribe to. It is fired whenever an error occurs. This event has properties to tell you the exception class that was raised and lets you specify a new text message for the error. Table 14-10 lists the properties for the ExceptionEventArgs() type.

Table 14-10. Properties of the ExceptionEventArgs event type.

Property	Description
Exception	The class of exception that was raised.
Handled	Set to True if you do not want to the error triggered.
UserData	Overrides the error message.

Unfortunately, the HandleException() event is not available if you are using the ReportDocument class. If you are printing a report directly with the ReportDocument class, there is no error-related event that you subscribe to. You have to use the standard Try-Catch error handling statements that you use for the rest of your application.

```
Try
  Dim myReport As New CrystalReport1()
  myReport.PrintToPrinter(1, False, 0, 0)
Catch
  ... do error handling here
End Try
```