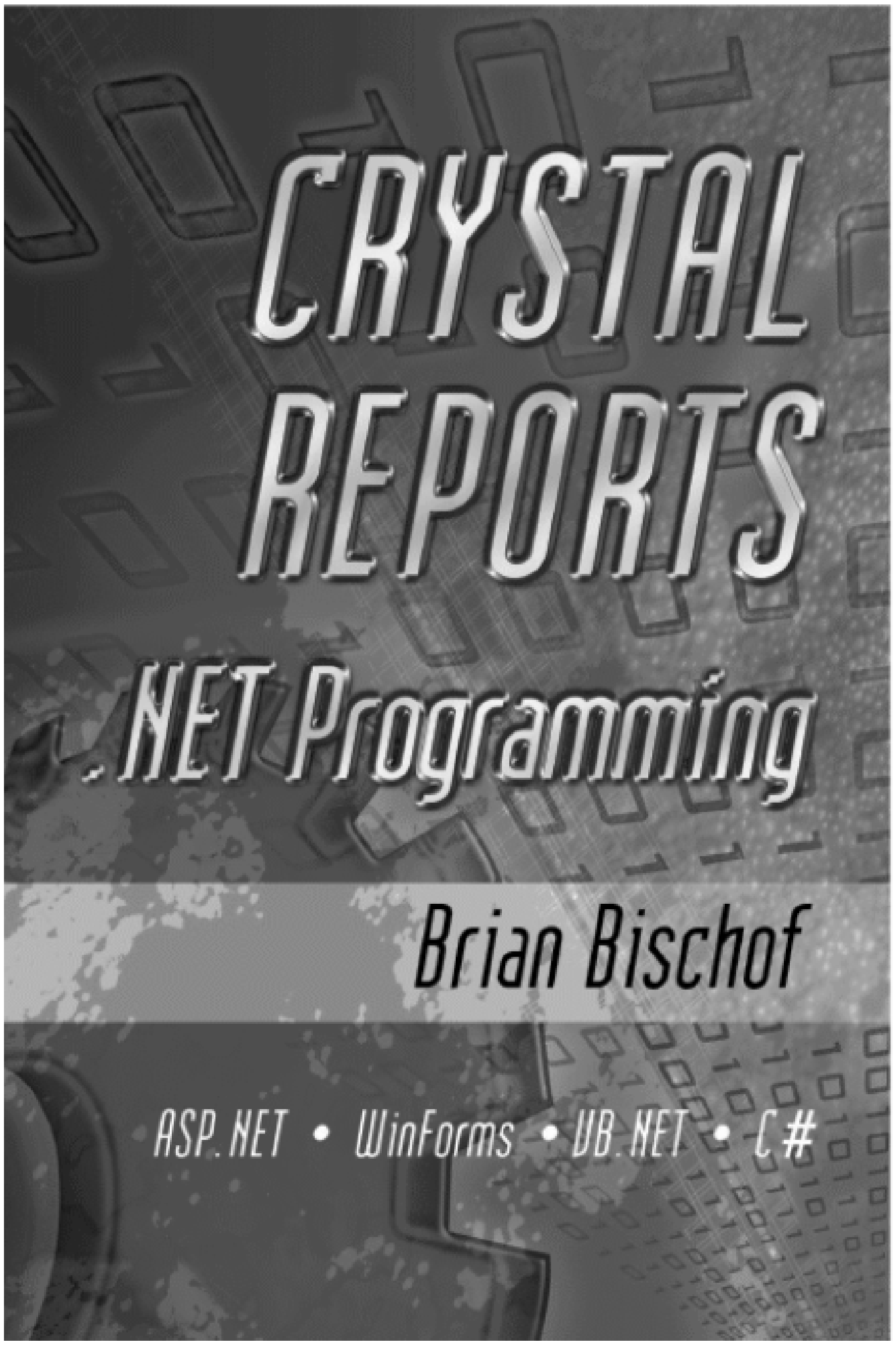
Excerpts from the book

Crystal Reports .NET Programming

By Brian Bischof
Copyright 2004



This free ebook gets you started in the right direction for learning how to program with Crystal Reports for .NET. The chapters in this book are direct excerpts from the book Crystal Reports .NET Programming (ISBN 0-9749536-5-2) available April 1st, 2004. This gives you a chance to not only learn a lot about Crystal Reports .NET for free, but also determine if the hardcopy version of the book is something that would benefit you. The complete table of contents of the hardcopy version is listed below. The remaining chapters of this ebook are excerpts of selected chapters. For more information see 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 afterthought. 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.

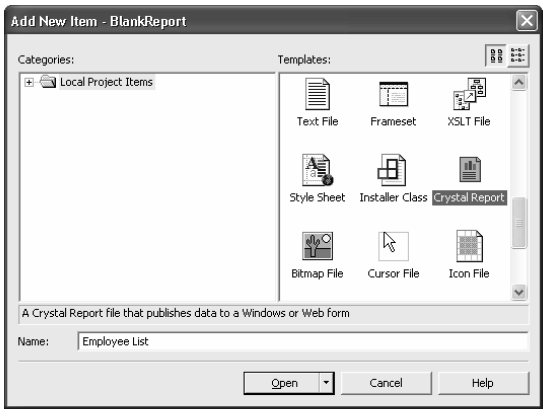
Over the years Microsoft has been including Crystal Reports with each version of Visual Basic. With version 6 they even wrote a Data Report component that was supposed to be a replacement for Crystal Reports. But it failed miserably.

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.



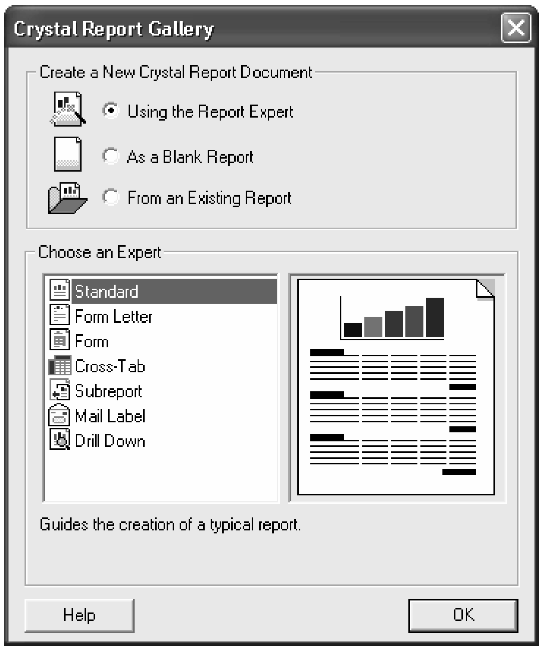
A Crystal Report file that publishes data to a Windows or Web form

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.

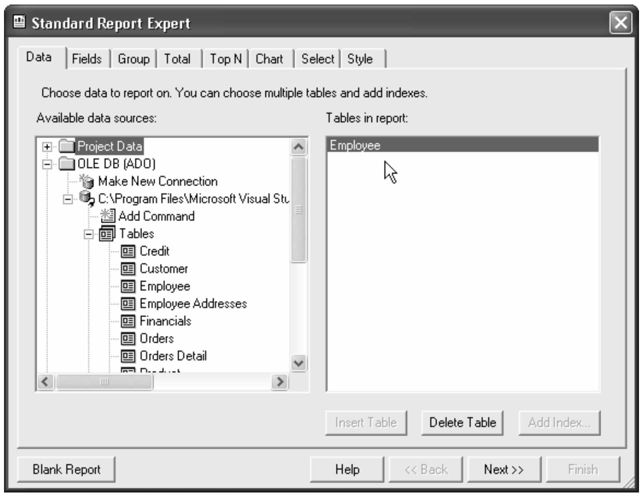


Guides the creation of a typical report.

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.

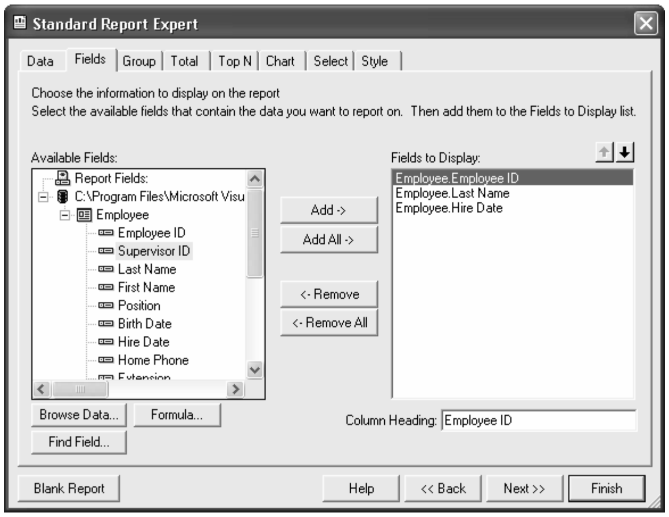
You are back at the Data tab of the expert. Click on the Tables node to expand and double-click on the Employee table name (or drag and drop it) to move it to the window titled Tables In Report. Click Next.



Choose data to report on. You can choose multiple tables and add indexes.

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.



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

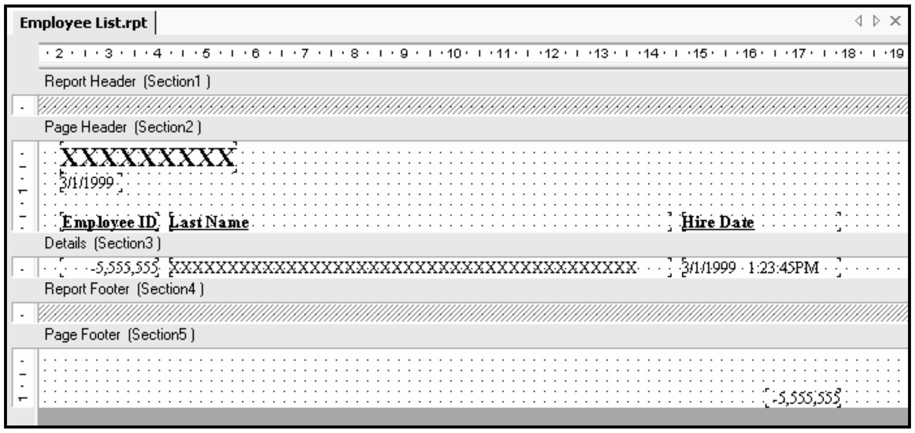
Fields to Display:
- Employee.Employee ID
- Employee.Last Name
- Employee.Hire Date

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.



Report Header (Section1)
Page Header (Section2)
XXXXXXXXX
3/1/1999
Employee ID Last Name Hire Date
Details (Section3)
-5,555,555 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 3/1/1999 1:23:45PM
Report Footer (Section4)
Page Footer (Section5)
-5,555,555

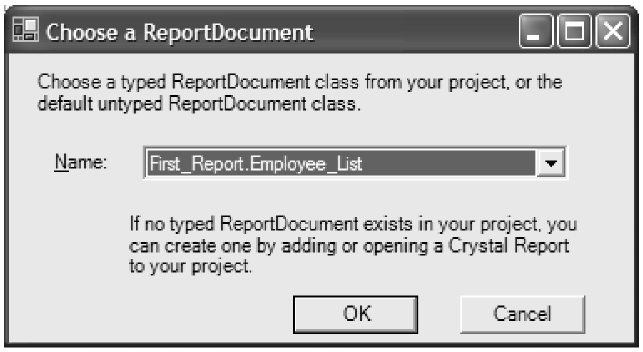
Figure 1-5. Employee List report file.

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.

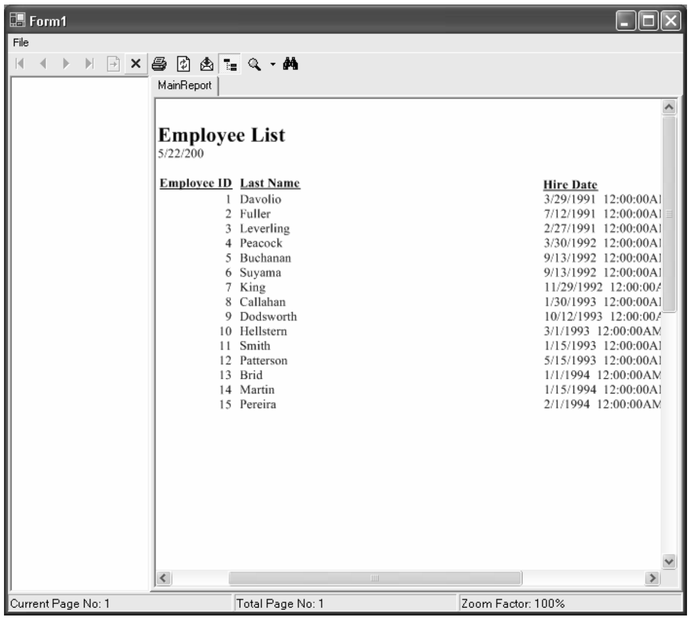
Figure 1-6. The ReportDocument dialog box.

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

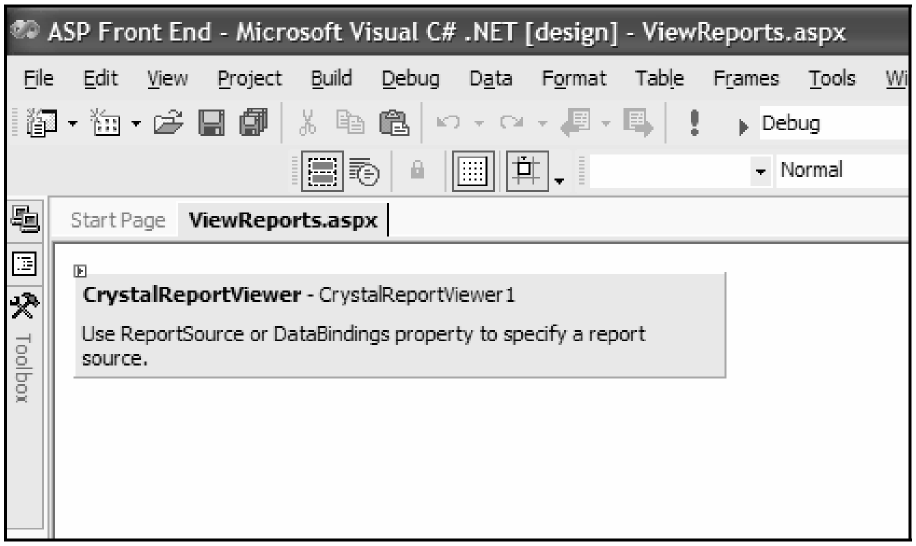
Employee ID	Last Name	Hire Date
1	Davolio	3/29/1991 12:00:00AI
2	Fuller	7/12/1991 12:00:00AI
3	Leverling	2/27/1991 12:00:00AI
4	Peacock	3/30/1992 12:00:00AI
5	Buchanan	9/13/1992 12:00:00AI
6	Suyama	9/13/1992 12:00:00AI
7	King	11/29/1992 12:00:00A
8	Callahan	1/30/1993 12:00:00AI
9	Dodsworth	10/12/1993 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.

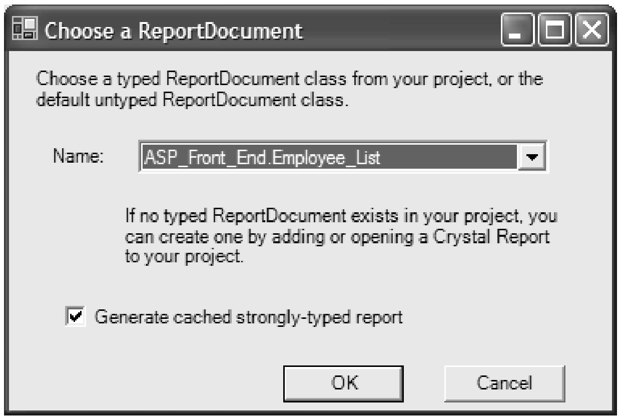


CrystalReportViewer - CrystalReportViewer1
Use ReportSource or DataBindings property to specify a report source.

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

Name: ASP_Front_End.Employee_List

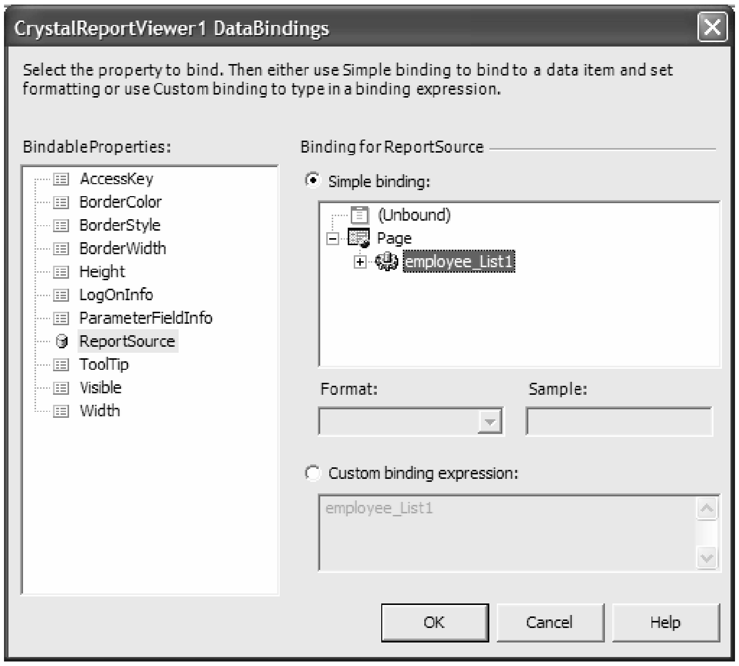
If no typed ReportDocument exists in your project, you can create one by adding or opening a Crystal Report to your project.

☑ 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.



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.

Binding for ReportSource

● Simple binding: employee_List1

○ Custom binding expression: employee_List1

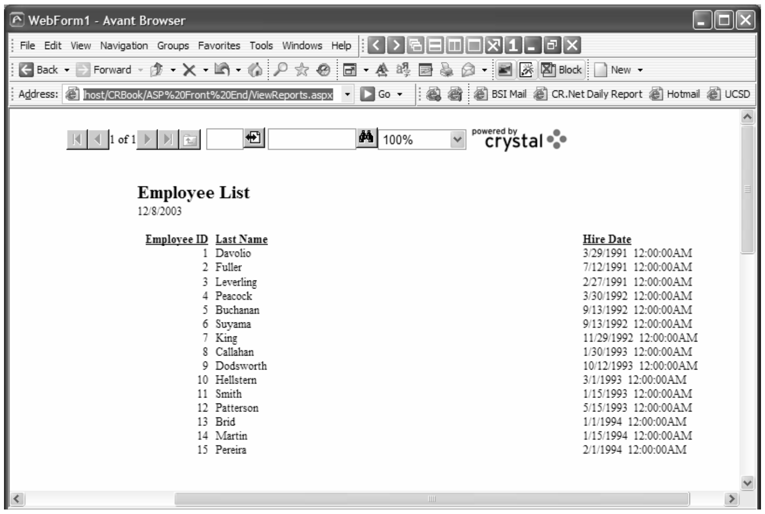
Click on the ReportSource property listed on the left. Expand the Page item in the Simple Binding window. It shows you the reports that were added to the page using the ReportDocument components. In this example, only the Employee List report is shown.

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.

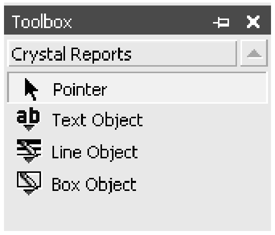


Figure 1-10. Crystal Report's toolbox.

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.

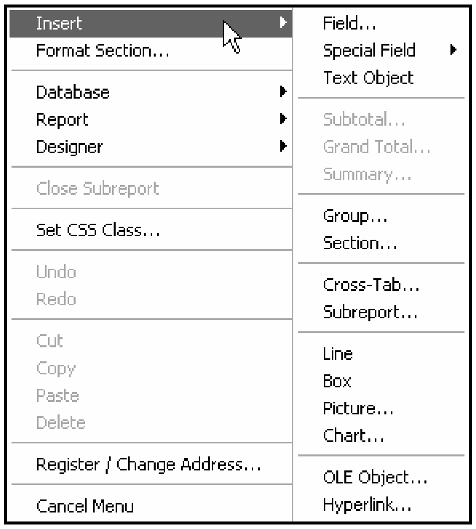


Figure 1-11. Crystal Report's Insert menu.

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.

2

Creating Reports

Since Crystal Reports is fully integrated with Visual Studio .NET, you may be tempted to think that creating a new report is a simple matter that only takes a minute or two. While this is true after you have a little experience, creating a report the first time can be confusing if you don't follow the proper steps. This chapter shows you what steps are required and how to do them, providing you with a foundation for all chapters to follow. After writing a couple of reports, these steps become second nature. There are two parts to learn: creating a report and printing a report.

Creating a Report

Reports are files that must be created with the Visual Studio IDE. This requires opening a project and building the files within it. Adding a report to a project involves 5 steps. Each step has different options that you need to consider before implementing it.

Table 2-1. Steps for writing a report.

Step	Description
1. Creating a new report file	From the menu, select Project/Add New Item. Follow the prompts to select the report type.
2. Running the report experts	Use the different report experts to identify the tables and fields to print and get a good start in the right direction. This is optional.
3. Setting the designer's defaults	Make sure your development environment is the way you want it before working on your report.
4. Adding report objects to the report	Add the different objects that your report needs. These objects consist of textboxes, fields, etc.
5. Formatting the objects	Set the properties of the objects so that they are formatted properly. These properties consists of fonts, sizes, etc.

Before you add a report to your project, you need to know what your reporting goals are, and how the report will help fulfill these goals for your application. None of the options presented here are complex, you just need to be aware of what they are in advance. The following sections explain each step in detail and tell you what you need to consider when doing each one. The last section of the chapter, Coding Samples, pulls together everything you learned in this chapter and shows sample code of how to print and preview reports.

Creating a New Report Class

To create and print a report, you need to create a new report class for your project. Once the class is created, you can modify it using the report designer and then call it from your application.

Creating a new report class is done two different ways. The first is to select the menu options Project | Add New Component. The other way is to right-click on your project name in the Solutions Explorer window and select Add | Add New Component. Both of these methods open the Add New Item dialog. Scroll down near the bottom to select Crystal Report, type in a report name, and click Open.

The Crystal Report Gallery dialog box opens (see Figure 2-1). You are given the option of using the report expert, creating a blank report, or creating a report using an existing report. If you choose the Report expert option, you can tell it the type of report expert to use in the list box below the options. If you choose to start with a blank report, the gallery goes away and a new report is created. This report gives you the five basic report sections and each is empty. Add report objects to the proper sections and format them to build your report.

You can also create a new report based off an existing report by selecting Project | Add Existing Item. This skips the Gallery dialog box and immediately shows you the Open File dialog box. A copy of the report is saved in the same folder as your application. All changes are made to the local copy and do not effect the original report.

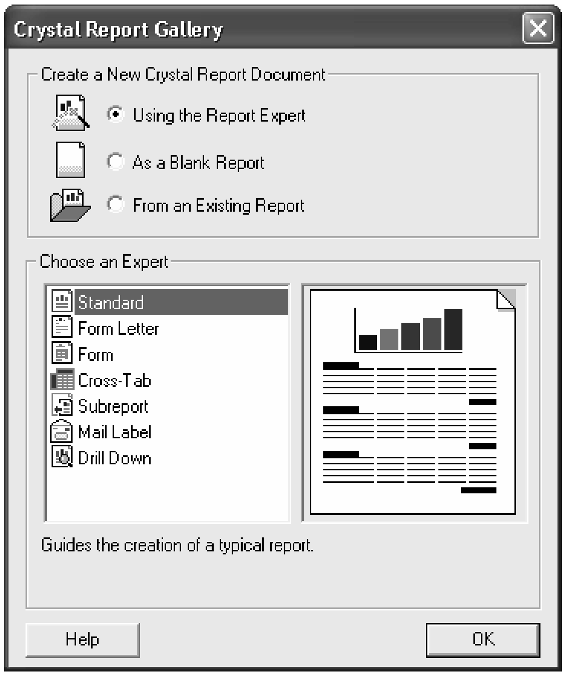


Figure 2-1. The Report Gallery dialog box.

Using the Report Experts

Depending upon your needs, creating reports can be a simple or complicated process. For example, it is very easy to print mailing labels from an address database or a form letter to a list of subscribers. On the other hand, it can be very complicated to write a report that uses multiple sub-reports which are based off of user entered parameters. Fortunately, the report experts that come with Crystal Reports, often referred to as "Wizards" in other applications, make it easy to quickly produce a variety of reports. The experts are useful for writing complex reports because they give you a way to quickly make a professional looking report template that you can customize for your specific needs.

Crystal Reports uses different experts to create seven types of reports: Standard, Form Letter, Form, Cross-tab, Sub-report, Mailing Label, and Drill-Down.

Since experts are designed to be easy to use, you might be wondering why this chapter needs to explain them. The problem with experts is that they are supposed to be simple to use, but often a tool that is simple is generally not very useful. As a result, today's applications are designed so that not only can you quickly answer the questions presented and create the final product, but you can also click on various buttons and checkboxes to add advanced functionality. This gives you the best of both worlds: a simple to use interface with extra features for advanced users.

Each expert consists of a multi-tabbed dialog box. Many of the tabs on each expert are used elsewhere within the main functionality of Crystal Reports. This chapter summarizes each tab and the details are covered in later chapters.

Using the Tabbed Dialog Boxes

Each report expert uses a combination of different tabbed dialog boxes to question you about how to build your report. Each expert presents a slightly different combination of these tabs. This section describes how to use each tab and explains any aspects of it that may not be obvious. The report expert tabs are called Data, Links, Fields, Group, Total, TopN, Chart, and Select.

The Data Tab

The Data tab is the first dialog box presented after you select which type of report you want. It lets you select the database and tables that store your data. The database can be a standard data source such as SQL Server or a non-standard data source such as an Excel spreadsheet.

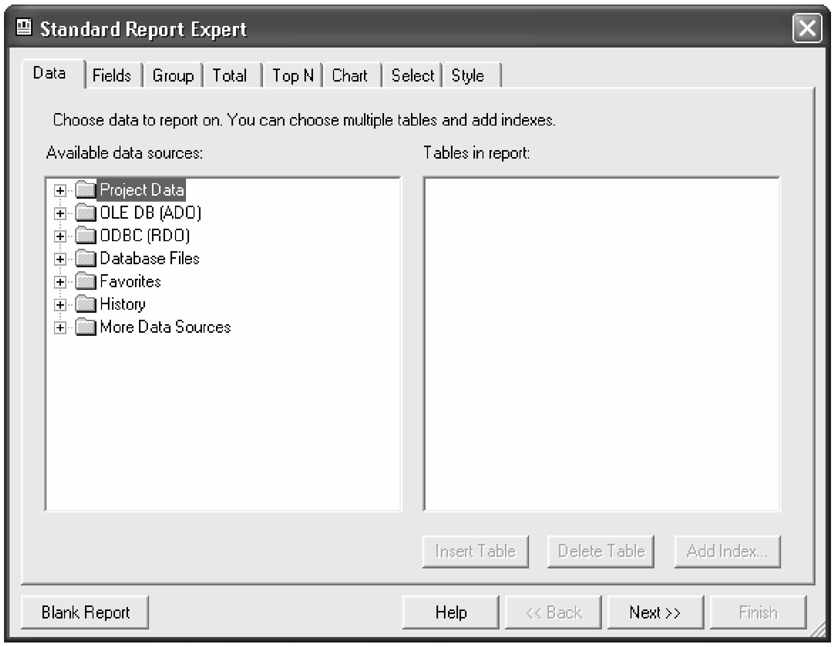
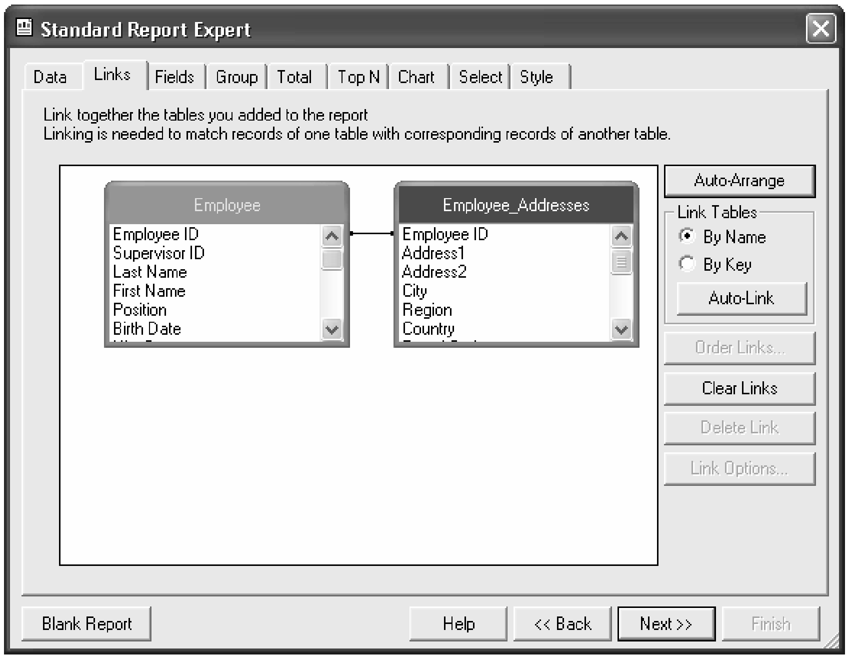

Choose data to report on. You can choose multiple tables and add indexes.

Figure 2-2. The Data tab of the report experts.

The Links Tab

Reports that use two or more tables need to have the tables linked so that data can be pulled from both of them. The Links tab lets you set the fields for creating relationships between the tables. Crystal Reports automatically attempts to link the tables together by using common field names. If two tables have a field with the same name and the data types are compatible, then Crystal will link them using this field.



Link together the tables you added to the report
Linking is needed to match records of one table with corresponding records of another table.

Employee: Employee ID, Supervisor ID, Last Name, First Name, Position, Birth Date

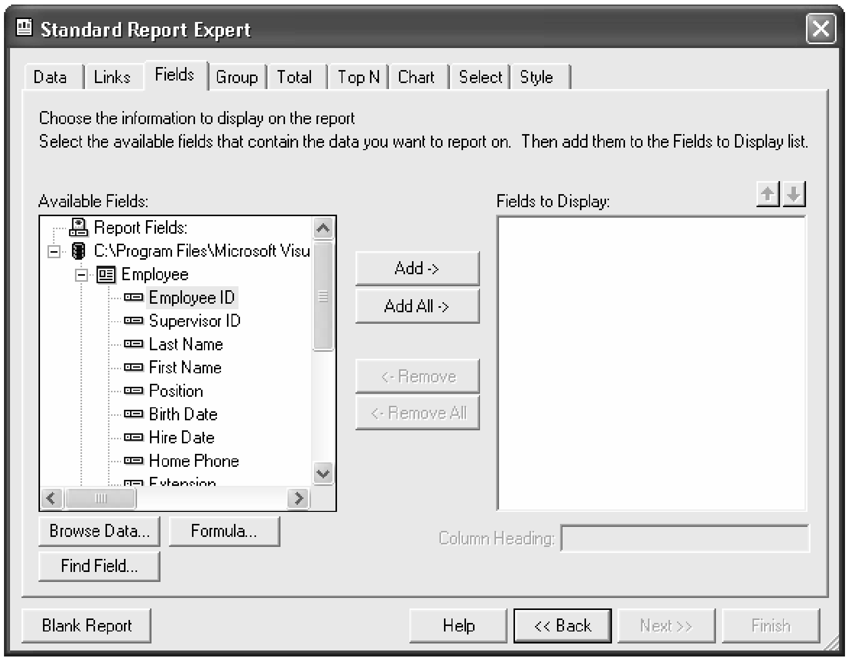
Employee_Addresses: Employee ID, Address1, Address2, City, Region, Country

Link Tables: [✓ By Name] By Key

Figure 2-3. The Links tab of the report experts.

The Fields Tab

After selecting which tables you want to use for your report, the Fields tab allows you to select which fields will be shown. . Adding and deleting fields is done in the standard manner of dragging and dropping them between windows or selecting a field and clicking on the appropriate arrow button.



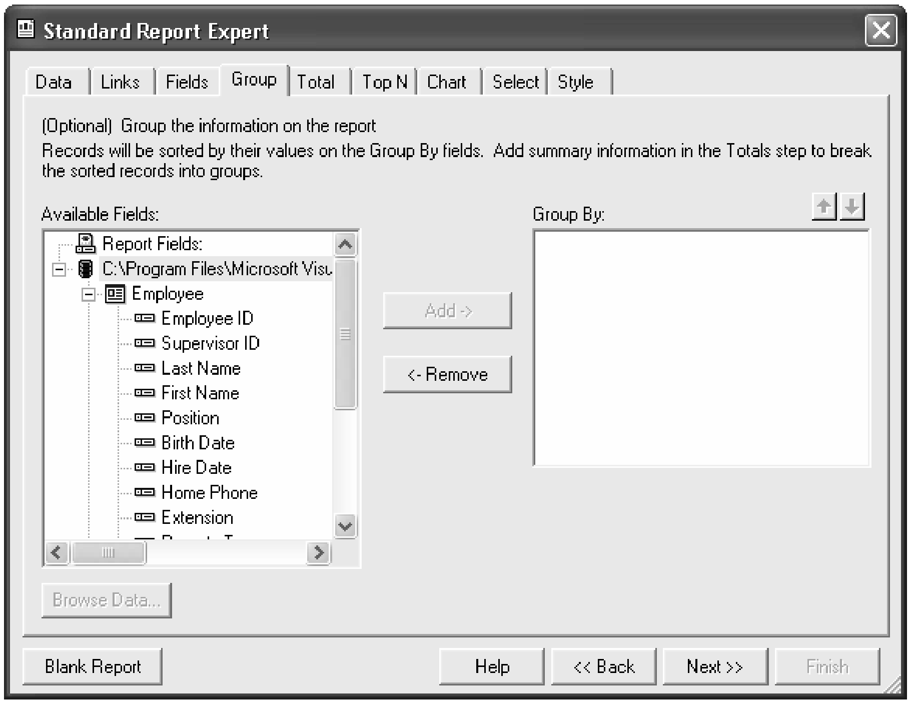
Choose the information to display on the report
Select the available fields that contain the data you want to report on. Then add them to the Fields to Display list.

Figure 2-4. The Fields tab of the report experts.

After you add all the necessary fields to your report, you are free to reorder them by using the arrow buttons above the window. Select the field to move and click the up or down arrow to reposition it.

The Group Tab

Some reports have so much data on them that they can be a little hard to understand. When reports are dozens or even hundreds of pages long, you need to organize the information in a way that makes it easier to absorb the data in smaller pieces. You do this by creating groups within the report. Groups can be based on many things. Some examples are grouping on months of the year or the names of the different branch offices for a company.

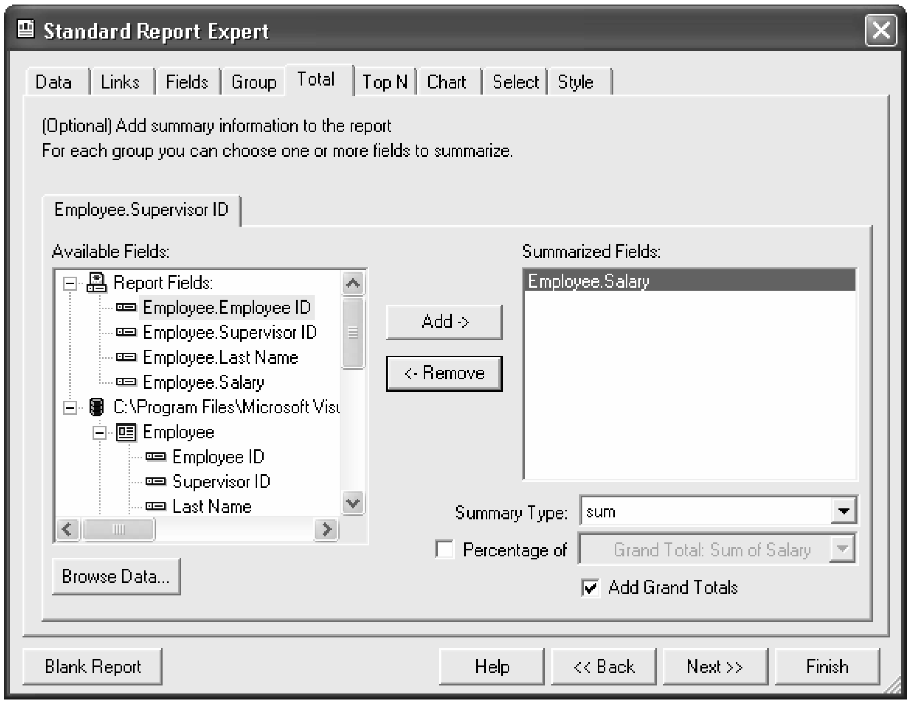


(Optional) Group the information on the report
Records will be sorted by their values on the Group By fields. Add summary information in the Totals step to break the sorted records into groups.

Figure 2-5. The Groups tab of the Report Expert.

The Total Tab

It is very common for reports to calculate sub-totals and other summary calculations on the numeric fields. The Total tab, shown in Figure 2-6, is where you define the summary calculations for the different fields. The left listbox shows all the available fields. The right listbox shows the fields that will have summary functions calculated for them.



(Optional) Add summary information to the report
For each group you can choose one or more fields to summarize.

Summarized Fields: Employee.Salary

Summary Type: sum

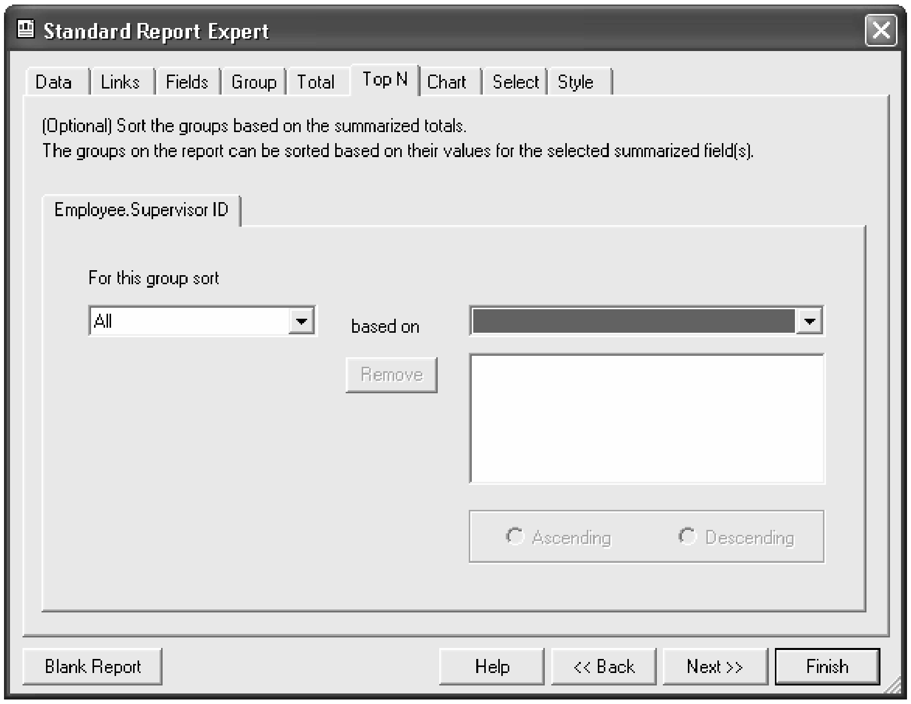
☐ Percentage of: Grand Total: Sum of Salary

☑ Add Grand Totals

Figure 2-6. The Total tab of the Report Expert.

The Top N Tab

When adding a group to a report, you probably assume that every record within the group will be displayed. In most circumstances this is the case. However, you can tell Crystal Reports to only display a certain number of records based upon their rank. For example, you could have a top salesperson report where you show the top 10 salespeople in your office. You could also have another report that shows the worst 10 salespeople. The Top N tab lets you do this by sorting the groups based on a summary field



(Optional) Sort the groups based on the summarized totals.
The groups on the report can be sorted based on their values for the selected summarized field(s).

For this group sort: All

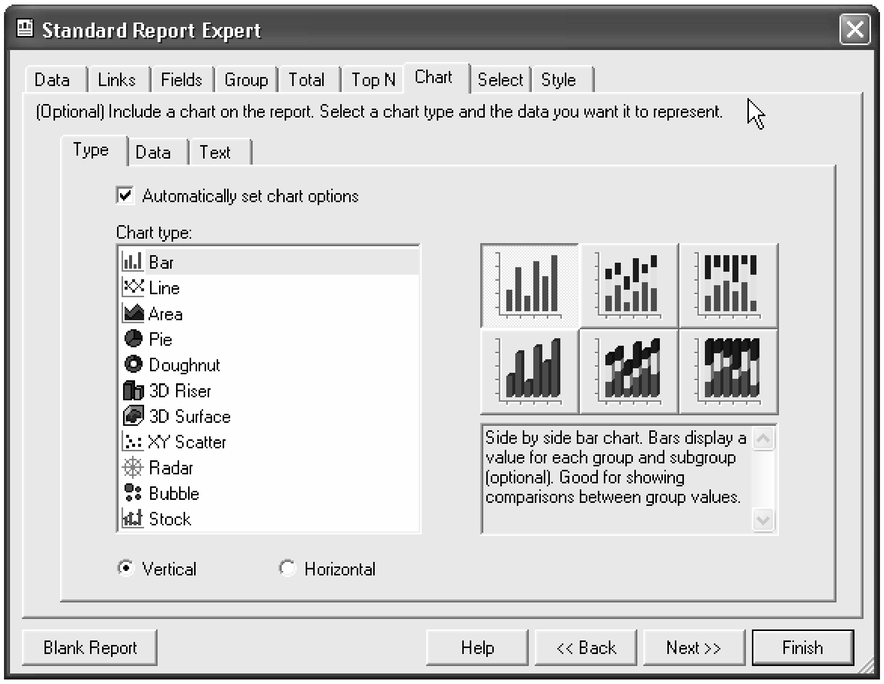
based on

Ascending Descending

Figure 2-7. The Top N tab from the Report Expert.

The Chart Tab

This tab lets you add a chart to your report. You can also set a variety of options for what type of chart to display and how to display it.



(Optional) Include a chart on the report. Select a chart type and the data you want it to represent.

[✓] Automatically set chart options

Chart type:
- Bar
- Line
- Area
- Pie
- Doughnut
- 3D Riser
- 3D Surface
- XY Scatter
- Radar
- Bubble
- Stock

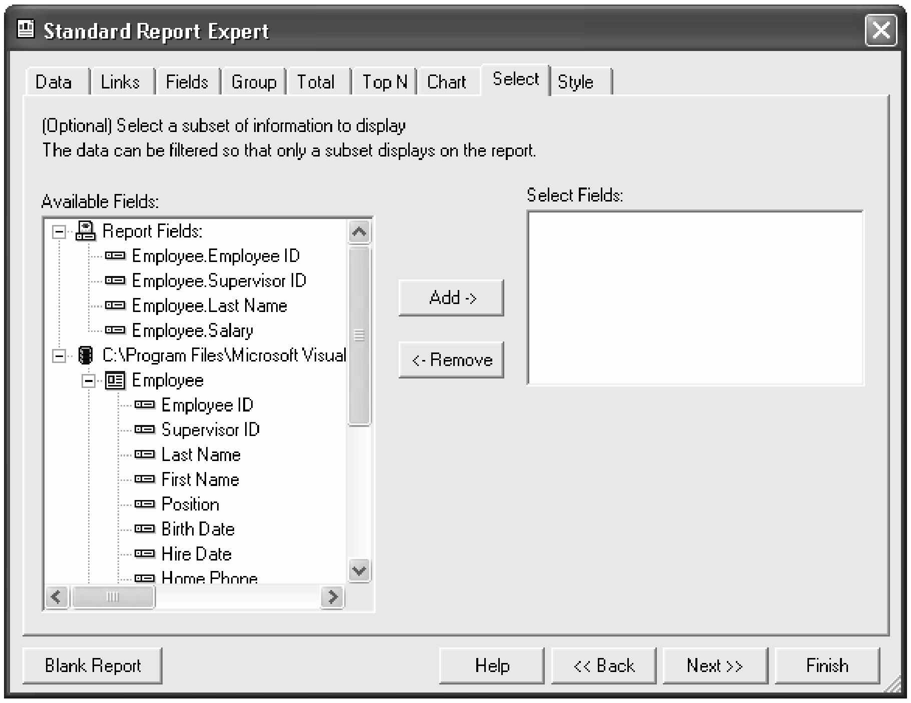
Side by side bar chart. Bars display a value for each group and subgroup (optional). Good for showing comparisons between group values.

[✓ Vertical] Horizontal

Figure 2-8. The Chart tab from the Report Expert.

The Select Tab

Some reports need to only show a sub-set of all the data that is in a recordset. For example, a financial report may only show the corporate data for a single quarter or a range of quarters. To filter out certain data so that you limit how much information is shown on a report, use the Select tab shown in Figure 2-9.



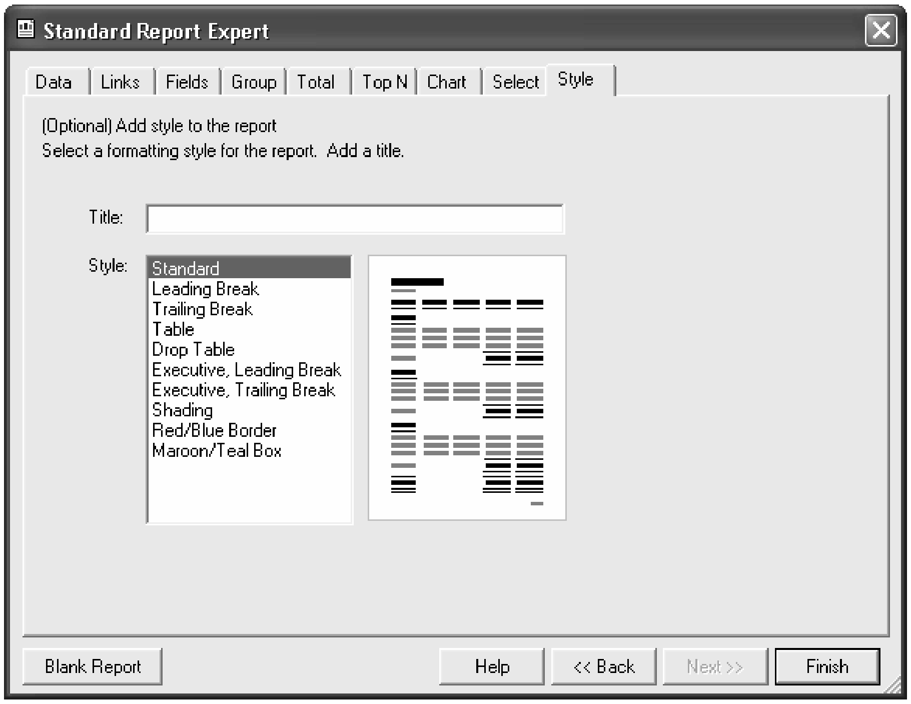
(Optional) Select a subset of information to display
The data can be filtered so that only a subset displays on the report.

Report Fields:
Employee.Employee ID
Employee.Supervisor ID
Employee.Last Name
Employee.Salary
C:\Program Files\Microsoft Visual
Employee
Employee ID
Supervisor ID
Last Name
First Name
Position
Birth Date
Hire Date
Home Phone

Figure 2-9. The Select tab from the Report Expert.

The Style Tab

After specifying the report details, use the Style tab to format everything so that the report has a professional look to it. This tab, shown in Figure 2-10, lists 10 different pre-determined formats that can be applied to your report. These styles make it easy for you to take some raw data and spice it up enough to make everyone think you worked really hard on this report!



(Optional) Add style to the report
Select a formatting style for the report. Add a title.

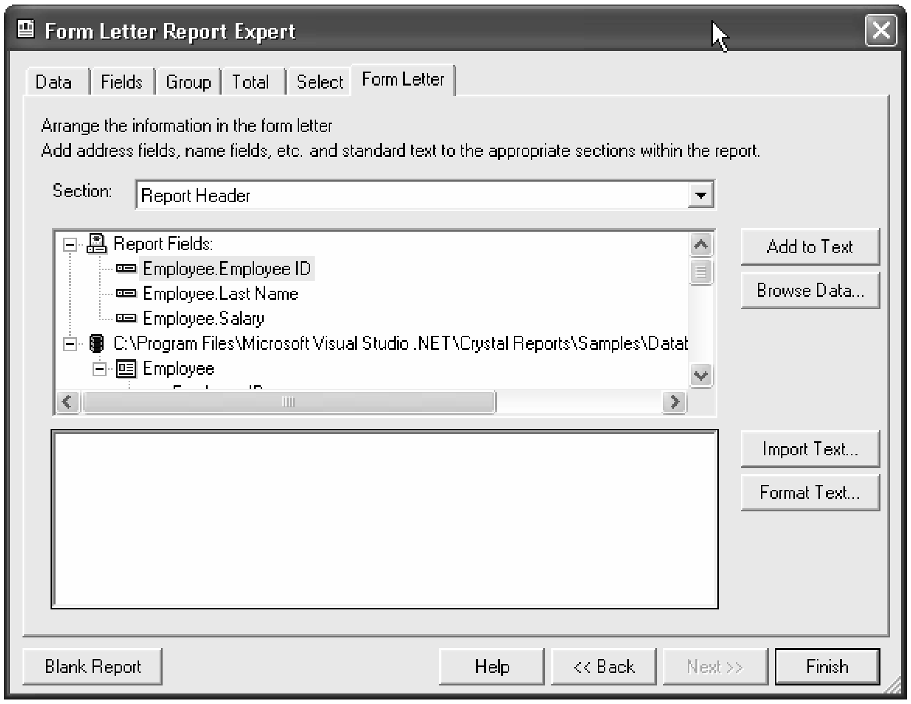
Title:

Style:
- Standard
- Leading Break
- Trailing Break
- Table
- Drop Table
- Executive, Leading Break
- Executive, Trailing Break
- Shading
- Red/Blue Border
- Maroon/Teal Box

Figure 2-10. The Style tab from the Report Expert..

The Form Letter Tab

This tab is only available when using the Form Letter Expert. It lets you write the entire form letter within this tab. See Figure 2-11. At the top is a drop-down box that lists the different sections of the report. For example, it lists the Report Header, Page Header, any group header sections and the Detail section. Select the section you want to write and type the text in the scrollable textbox below it. You can also insert formulas or fields from the data sources in this textbox and the expert keeps track of it all. When you are finished and you close the expert, it adds a large text box to your report for each section and fills in the information you entered.



Arrange the information in the form letter
Add address fields, name fields, etc. and standard text to the appropriate sections within the report.

Section: Report Header

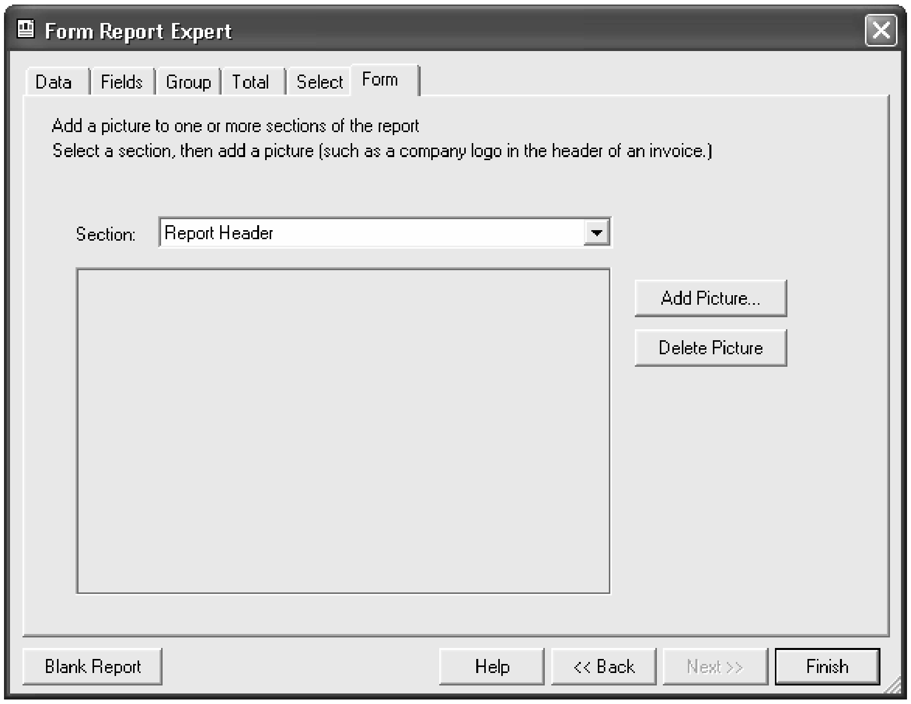
Report Fields:
Employee.Employee ID
Employee.Last Name
Employee.Salary
C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Datat
Employee

Figure 2-11. The Form Letter tab from the Report Expert.

The Form Tab

The Form tab, shown in Figure 2-12, is only available when using the Form Expert. It gives you a way to easily add a picture or logo to the sections of a report. Often when you add a picture using the expert, it might not be sized properly and could look distorted. Don't worry about that because it will look fine on the report and if you need to, you can always change its properties in design mode.

The Form tab also makes it easy to write reports that print onto standardized forms. Scan in the form that you want to print onto. Then use an image editor to crop it into the appropriate sections: Header, Detail, Footer, etc. Insert each section into the proper section of your report. Add the fields so that they line up exactly with the scanned image. When you are finished adding all the fields, delete the images. This leaves you with only the fields to print and they are lined up perfectly.


Add a picture to one or more sections of the report
Select a section, then add a picture (such as a company logo in the header of an invoice.)

Section: Report Header

Figure 2-12. The Form tab from the Report Expert..

The Cross-Tab and Customize Style Tabs

These tabs are only available when using the Cross-Tab Expert. They let you modify the cross-tab data for what appears in the rows and columns. Modifying these tabs is very involved and is explained in detail in Chapter 11.

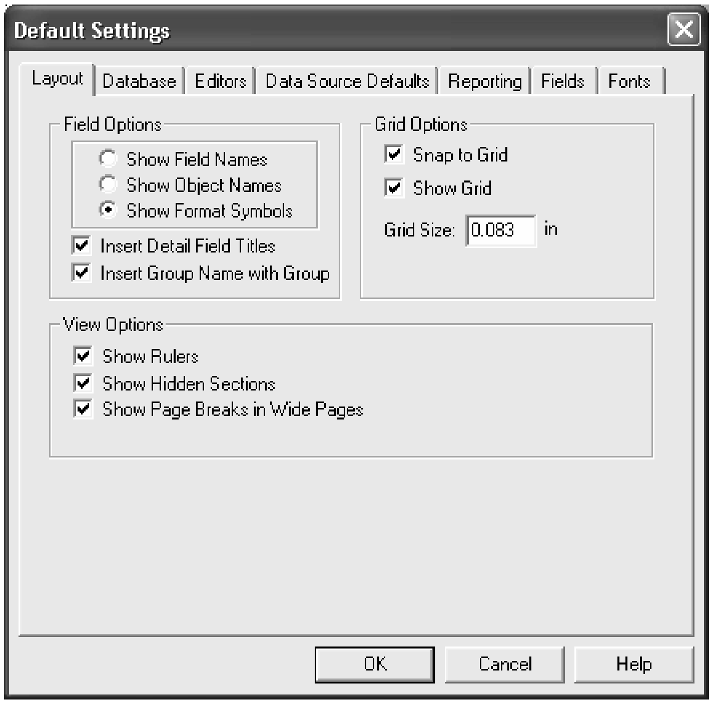
The Drill Tab

The Drill tab is only available when using the Drill-Down Expert. When creating drill-down reports, you allow the user to see a summarized view of their data. If they want to examine the data in more detail, they can expand a record to see more detail relating to the summarized information. The summarized fields are listed on this tab. When you click on the field's name, it will toggle it back and forth between "Show" and "Hide". This will alternate the default of whether it gets displayed or not.

Setting the Designer's Defaults

The report designer is where you will spend all your time creating reports and modifying them. There are many different default settings that you can set that control how you interact with the report designer as well as controlling how you access and display data on a report. Knowing what the different options are will not only make you more efficient, but it can also save you a lot of headaches. It is very likely that once you set these default values you will not come back to this step. However, when starting a new application, it would be beneficial to set the default values for how different report objects are displayed. For example, if you decide to always make group headers a certain font, then you can set that to be the default and you won't have to modify it every time.

To modify the default settings, right-click anywhere on the report and select Designer/Default Settings. The Default Settings dialog box appears. There are seven different tabs that control the default settings. Each tab affects different parts of the designer and report output.



Field Options
- Show Field Names
- Show Object Names
- [✓ Show Format Symbols]
- ☑ Insert Detail Field Titles
- ☑ Insert Group Name with Group

Grid Options
- ☑ Snap to Grid
- ☑ Show Grid
- Grid Size: 0.083 in

View Options
- ☑ Show Rulers
- ☑ Show Hidden Sections
- ☑ Show Page Breaks in Wide Pages

Figure 2-13. The Default Settings dialog box.

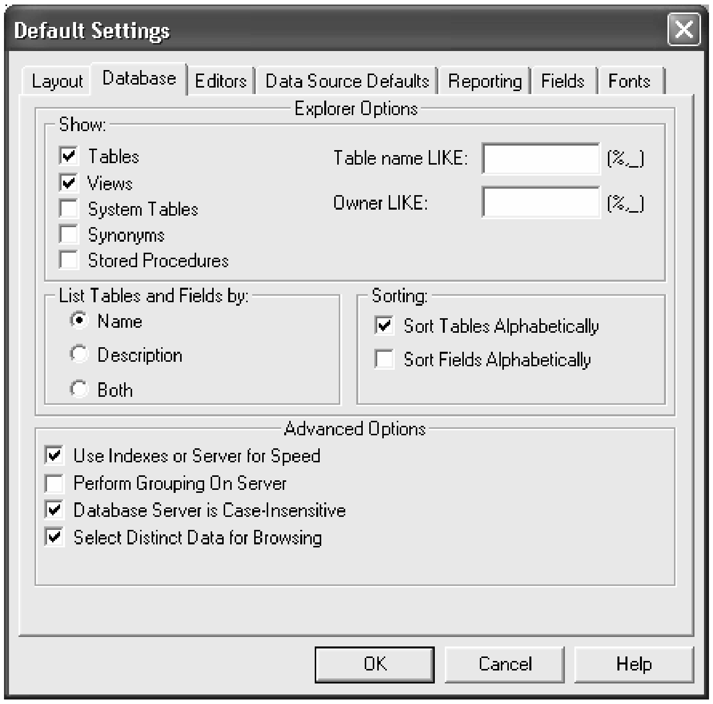
The Layout Tab

The Layout tab effects your interaction with the report designer. The Field Options frame sets how fields are shown on the designer. You have the option of showing the names of the fields that are being displayed, the names that were assigned to each object, or format symbols. Showing the format symbols is useful for seeing how the different numbers will be formatted and seeing the maximum width that a string can use.

The Grid Options frame will be discussed in the Adding, Resizing and Moving Controls section. It lets you turn the grid markers on and off and set whether objects must align with them. The View Options frame sets what is shown in design mode.

The Database Tab

The Database tab effects how items are displayed in the Field Explorer and it has settings for tweaking performance. See Figure 2-14. The settings for changing the Field Explorer are many. They consist of deciding what types of objects are to be displayed and how the items will be sorted.


Default Settings
Layout | Database | Editors | Data Source Defaults | Reporting | Fields | Fonts

Explorer Options

Show:
- [✓] Tables
- [✓] Views
- [ ] System Tables
- [ ] Synonyms
- [ ] Stored Procedures

Table name LIKE: (%,_)

Owner LIKE: (%,_)

List Tables and Fields by:
- (●) Name
- ( ) Description
- ( ) Both

Sorting:
- [✓] Sort Tables Alphabetically
- [ ] Sort Fields Alphabetically

Advanced Options
- [✓] Use Indexes or Server for Speed
- [ ] Perform Grouping On Server
- [✓] Database Server is Case-Insensitive
- [✓] Select Distinct Data for Browsing

Figure 2-14. The Database tab.

Performance related settings are listed in the Advanced Options frame. The setting to use indexes on the server is set by default. Using indexes on the server gives you better performance because the server is optimized for doing this. The other performance related setting is to perform grouping on the server. Just like performing indexes on the server is faster, so is having the server do the grouping. Unfortunately, you can only use this option for accessing SQL tables directly. It isn't available when reporting off queries.

The Editors Tab

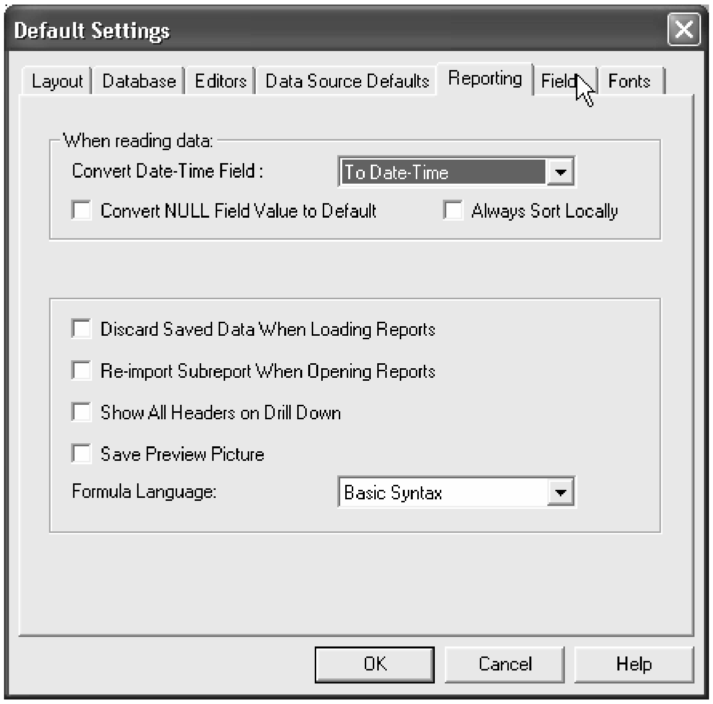
The Editors tab modifies those fonts that the Formula Editor and SQL Expression Editor use to display the programming code. You are free to customize the different programming elements to your heart's content.

The Data Source Defaults Tab

The Data Source tab lets you specify a default folder for where the database files are located. This is used with the Data tab of the Report Expert. When selecting Database Files it will display the Open dialog box and default to the folder you specified.

The Reporting Tab

The Reporting tab affects the output of your reports. See Figure 2-15. The top frame lets you set how data is read from the data source. You can automatically convert DateTime data to a String, a Date, or keep it as a DateTime data type. See Chapter 8 for a discussion of the date data types. Within this frame you can also convert NULL fields to their default value. This is very useful for ensuring that numeric fields are printed as a zero and not an empty field.



When reading data:

Convert Date-Time Field : To Date-Time

☐ Convert NULL Field Value to Default ☐ Always Sort Locally

☐ Discard Saved Data When Loading Reports

☐ Re-import Subreport When Opening Reports

☐ Show All Headers on Drill Down

☐ Save Preview Picture

Formula Language: Basic Syntax

Figure 2-15. The Reporting tab.

The bottom frame contains miscellaneous options. Reports can save their data so that they don't have to reload and process the records every time (this really improves printing speed). If you want your reports to discard the old data and always re-query the data source for the latest data, then turn the option on. You can also tell it to re-import subreports so that they are always refreshed when printing a report.

You can set whether drill-down reports will show the column names for the drill-down data. A preview picture (thumbnail) can be saved every time you run a report. The last option lets you set the default formula language as discussed in Chapter 8.

The Fields Tab

Every data type that can be displayed on a report has a default display format. For example, the default for displaying a number is to use two decimal places. The Fields tab lets you set the default formats for all the available data types (see Chapter 8 for a list of data types). By changing the default here, every new object that is added to your report after changing the default will use this setting. Any objects that were added before you made the change will not reflect the new format.

The Font Tab

Just as the Fields tab sets the default format for the different data types, the Font tab sets the default font for the different fields on a report. For example, you can make field titles appear in italics while group names appear in bold. Once again, these changes will only affect objects that are added after you make your changes.

Using the Report Objects

A report is very similar to a Windows form. Just like a form, the report is listed as a separate object under your project in the Solution Explorer window, there are various controls that can be added, and it is a class that has to be instantiated before using it. The objects that are added to a report are also similar to the ones you use when building forms. This section describes the different objects and how they are used. It also gives you a reference for the properties of each object. Every property isn't listed because that would require reprinting the MSDN documentation.[3] But the properties you'll need to use on a regular basis are highlighted here.

There are three controls in the report toolbox (see Figure 2-16). You can access these and many others by right-clicking on the report and selecting the Insert menu option (see Figure 2-17).

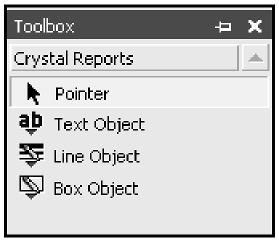


Figure 2-16. The control Toolbox.

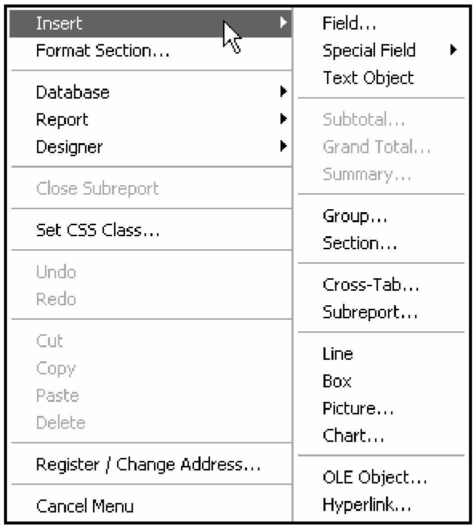


Figure 2-17. The Insert menu option.

Adding, Resizing and Moving Report Objects

The objects on a report work the same as the controls on a form. The objects that are listed in the toolbox can be added to the report by dragging and dropping them onto the proper section of the report. You can also double-click on them and they will be automatically added to the section that has the focus (its header bar will be blue while the other header bars will be gray). If you have an existing control that you want to reuse and it has already been formatted, then you can highlight it and copy and paste it. This creates a copy that is attached to your pointer and will move around as you move your mouse. When you have it positioned properly, click the mouse button to drop it there. You can also select multiple objects for copy and paste.

Selecting multiple objects is done by holding down the control or shift key and clicking on the individual controls. You can also draw a temporary window on the report and any controls that are included in the window get selected. Do this by holding down the mouse button and moving the mouse to enlarge the box. Let go of the mouse when the box is complete and the objects will be selected.

There is a strange behavior to be aware of when selecting multiple objects using the window technique. You can't draw a window if another object is already selected. You have to first click anywhere on the report to unselect the current object and then you can draw the box. For example, assume that you selected a textbox

[3] There are already too many .NET books that consider printing out the MSDN tables to be quality writing. I don't want this book to put in that group of trash.

object. You then decide that you really wanted to select multiple textboxes so you click elsewhere on the report and attempt to draw a window. Unfortunately, nothing will happen. It only results in the textbox getting unselected. You need to click the mouse again to start drawing the window.

You can only select multiple objects when they are compatible. For example, the box and line objects can be selected together, but the box object can't be selected with the text object.

Resizing a control is done by selecting it to give it the focus. Position the mouse over the sizing handles on any side and drag them. When resizing multiple objects, the sizing handle will only appear on the last control selected. As you resize the last control, its new size changes as you move your mouse. The other controls will not change until you release the mouse button. An option for resizing objects is to let Crystal adjust their size to be the same for each one. After selecting all the objects, right-click on one and select the Size menu option. From there you can choose Same Width, Same Height, or Same Size.

When moving objects on the report, it can be helpful to display the grid lines. This makes it easier to line up objects with each other. You have the option of making the objects snap to the grid lines. This means that when you move a control, its edge must be placed on a grid line. It can't be placed between grid lines. When you release the mouse, the object will automatically snap to the nearest grid line. Turning grid lines feature and the snap-to feature on and off is controlled by changing the designer properties. Right click on the report and select Designer/Default Settings.

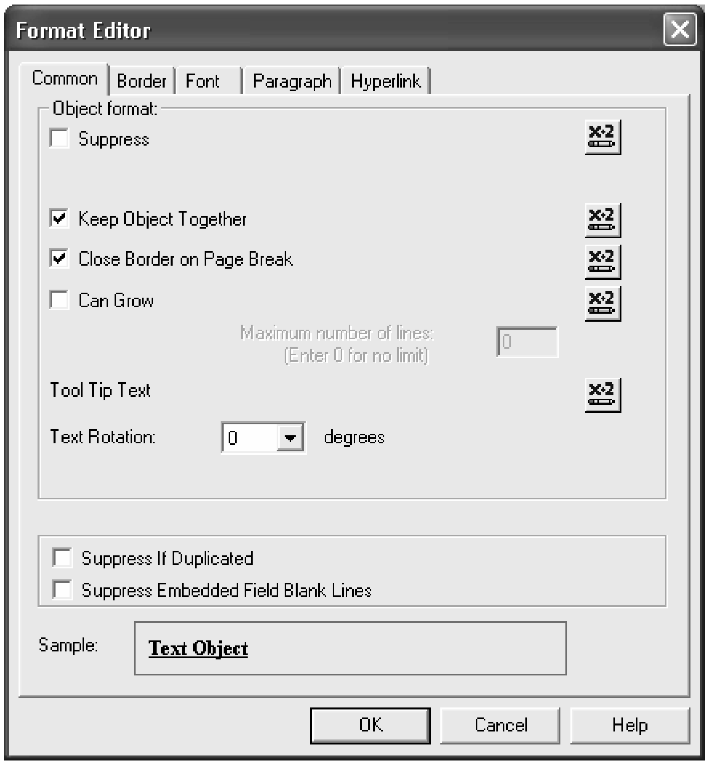
Turning on the snap-to grid lines option can be a blessing or a curse. There are two problems that can occur when this option is turned on, and they both seem to occur most frequently with reports that have been created with the report experts. The first problem is that when the report expert creates a report, it doesn't always place the controls on the grid lines. When you later want to rearrange the controls on the report, or add more controls, you can't get the new objects to line up because they are snapping to the grid lines. Since the wizard placed the objects between grid lines, they won't line up properly when they are on the grid lines. The second problem with using the snap-to grid lines option is that it can be impossible to move multiple objects and keep their spacing consistent. I call this the "rubber band effect". As you drag the objects across the report, some will move and others won't. But as you move your mouse further, the ones that haven't moved will now seem to snap into place and catch up with the other objects. If you move your mouse fairly quickly across the page they will appear to bounce around like they are being pulled by rubber bands. Unfortunately, you have no control over this and sometimes it can be impossible to make all the objects line up as you expected.

An easy way to align multiple objects is to use the Align menu option. After selecting the objects, right-click on one of them and select Align. The options to choose from are Tops, Middles, Bottoms, Baseline, Lefts, Centers, Rights, and ToGrid. Each of these options aligns multiple objects to a single object. The object that is used as the basis for alignment is the last object selected. You can identify the last object selected by seeing which one has the sizing handles on it.

Formatting Strings

String output can be formatted in a way similar to formatting a cell in a spreadsheet. You can modify its font and border. There are also formatting options that are specific to Crystal Reports. These options consist of suppressing the field, letting the width grow, rotating the text, paragraph specific formatting and hyperlinks. Almost every option on this dialog box can be set using formulas. This is discussed in more detail in Chapter 7. The formatting options are available for textbox objects, formulas, report fields, and special fields. Access the formatting dialog box by right-clicking on the field and selecting Format.

The first tab of the Format Editor dialog box is the Common tab (see Figure 2-18). The properties shown on this tab are common to most of the report objects available. These properties are described in Table 2-2.



Object format:
- [ ] Suppress
- [✓] Keep Object Together
- [✓] Close Border on Page Break
- [ ] Can Grow

Maximum number of lines: (Enter 0 for no limit) 0

Tool Tip Text

Text Rotation: 0 degrees

- [ ] Suppress If Duplicated
- [ ] Suppress Embedded Field Blank Lines

Sample: Text Object

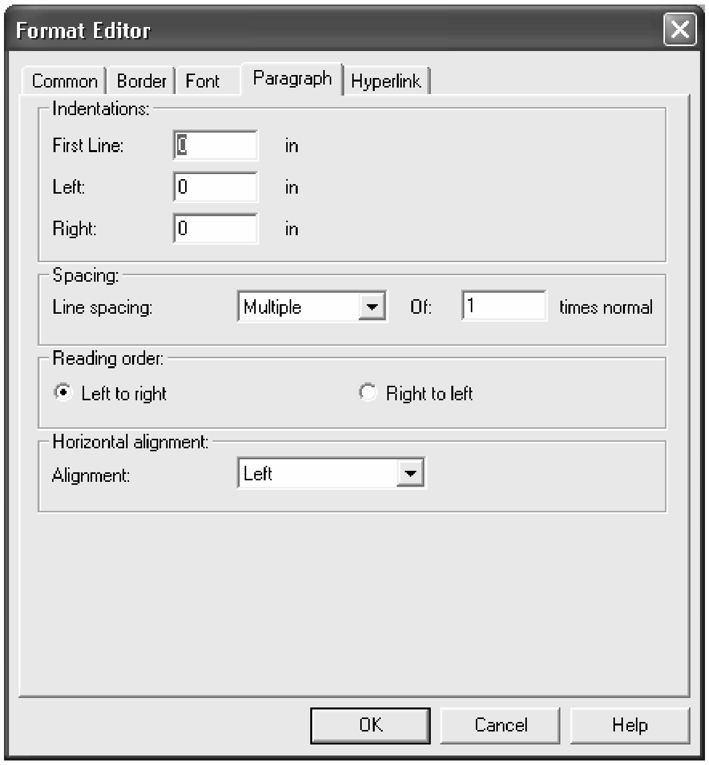
Figure 2-18. The Common tab.

Table 2-2. Properties of the Common Tab

Property	Description
Can Grow	Allow the field to expand if the object isn't big enough to hold the data. The field will only expand vertically and result in the height increasing. The width does not expand.
Close Border on Page Break	If a field has a border, and the field extends to another page, then this will close the border on the first page.
Horizontal Alignment	Sets the alignment to Left, Right, Center or Justified.
Keep Object Together	Do not let the object cross over into another page.
Suppress	Hides the object.
Suppress If Duplicated	Hide the object if it had the same value in the prior record.
Text Rotation	Rotate the text to a specified angle. A value of 0 is the default and the text displays horizontally. A value of 90 rotates it vertically upward. A value of 270 rotates it vertically downward.
Tool Tip Text	Use the formula editor to set a string that is displayed when the mouse hovers above the field.

The next two tabs on the Format dialog box are the Border tab and the Font tab. Both of these tabs are simplistic and don't have anything unusual in them. The Border tab lets you specify which sides should have a border and it also lets you can change the shading around the object. The Font tab has properties to change the font and use different effects such as strikethrough and underline.

The third tab of the dialog box is the Paragraph tab. This is useful when using multi-line objects and you want it to be formatted like a standard paragraph (see Figure 2-19). It lets you customize the indentation on the left and right sides. You can also specify a different indentation for the first line so that it is inset from the rest of the lines. You can also set the line spacing, the reading order (left to right or vice-versa), and specify if it uses plain text or text formatted as RTF.



Indentations:
First Line: 0 in
Left: 0 in
Right: 0 in
Spacing:
Line spacing: Multiple Of: 1 times normal
Reading order:
● Left to right ○ Right to left
Horizontal alignment:
Alignment: Left

Figure 2-19. The Paragraph tab.

The last tab is the Hyperlink tab. This is used for fields that store text to be used as a hyperlink to a website, an email address, a file, or another report. When you select what type of hyperlink it is, the middle frame changes so that you can enter the appropriate type of link. You can either hardcode the hyperlink or use the formula editor to make it dynamic depending upon what is in the field. Depending on the type of object, the hyperlink tab might be unavailable.

The Text Object

The text object is used to display text, database fields, and special report fields. Each text object can display one of these or a combination of all three. After adding a text object to the report, click on it to edit it.

When writing form letters, using the textbox object with a combination of text and database fields is very helpful. Text in form letters is different than standard reports because it doesn't follow the standard format of showing individual rows and columns. It is displayed in paragraph format with a single space between each word/number. The textbox makes this possible by automatically trimming the spaces around each word and number. Thus, you can insert database fields in with standard text. Any extra spaces around the database fields are not be shown. If you have ever programmed in HTML, then you are familiar with this concept. Within HTML, no matter how many spaces you put between words, they will be separated by only one space.

The properties that apply to the Textbox object are covered in the section Formatting Text Objects.

The Field Object

The standard field object displays data from a table or a formula. The field object is added to your report by dragging and dropping it from the Field Explorer window onto your report. The Field Explorer window is normally docked on the toolbar to the left side of the screen. If you don't have it there, you can pull it up by clicking on the menu items and selecting View/Other Windows/Document Outline. The properties that apply to the field object are covered in the Formatting Text Objects section.

The Line Object

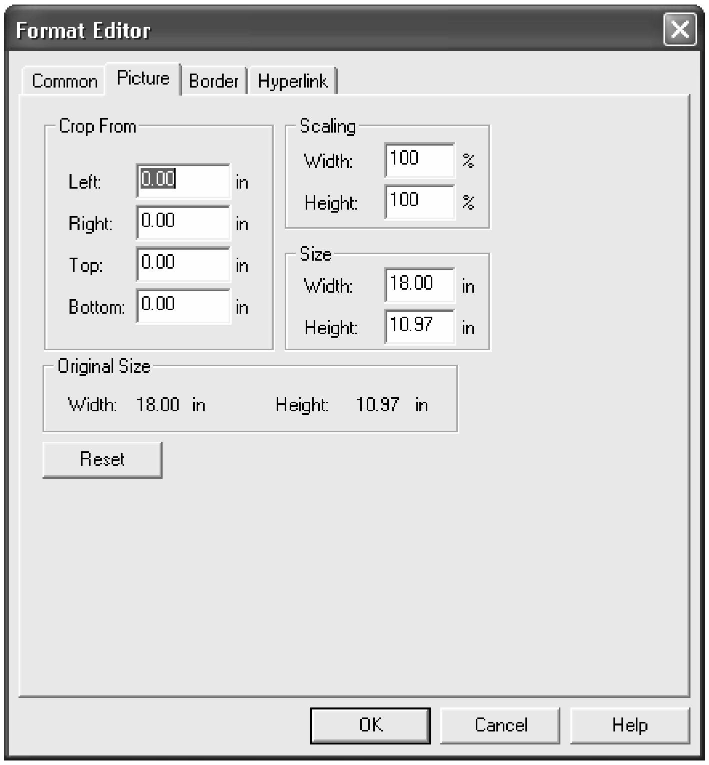
The line object does exactly what you expect. It draws a line. There isn't a whole lot you can do with this except change its color, the width and its style (single line, dashed line, etc.) It does have one interesting feature that solves a common problem with line objects. The problem occurs when you draw a vertical line on a detail section. There are times when the detail section has a field that can grow down the report and make the section longer than expected. You expected the line to be unbroken down the report and now that isn't the case because the line is too short. To fix this, set the property ExtendToBottomOfSection to True. This insures that no matter how short your line is, the end point will be the bottom of the section. If it is a horizontal line, this always move to the bottom of the section.

The Box Object

Like the Line object, the Box object isn't too exciting. You can change its color, width and style. One nice feature about it is that you can change its properties to round the edges. Depending on how you set the properties, you can make it elliptical or even turn it into a circle. The properties that affect this are CornerEllipseHeight and CornerEllipseWidth. Rather than modify these directly using the Properties window at the bottom right hand corner of the screen, it is much easier to use the Format dialog box. The second tab is the Rounding tab. It has a picture of what the box looks like and below it is a slider. As you move the slider from the left to the right, it increases the curvature of the edges. When the slider is at the far right, the box has turned into a circle. Once you click OK, the dialog box modifies the corner ellipse properties for you.

The Picture Object

The picture object is used for displaying the following image file formats: BMP, JPG, TIFF, and PNG. It doesn’t display GIF files. The formatting options are similar to the other controls in that it has the tabs Common, Border, and Hyperlink with similar functionality. There is a Picture tab, shown in Figure 2-20, that lets you resize, scale and crop the image. If at any point you feel that you resized or scaled it improperly and you want to restore it back to its original size, click the Reset button.


Format Editor

Common | Picture | Border | Hyperlink

Crop From
Left: 0.00 in
Right: 0.00 in
Top: 0.00 in
Bottom: 0.00 in

Scaling
Width: 100 %
Height: 100 %

Size
Width: 18.00 in
Height: 10.97 in

Original Size
Width: 18.00 in   Height: 10.97 in

Figure 2-20. The Picture tab of the Picture object's dialog box.

The Chart Object

The Chart object lets you add a chart to your report. There are a variety of options to choose from so that you can customize however you need to. There is so much to cover with this object that it was given a separate chapter. See Chapter 10 for more information.

The OLE Object

The OLE object lets you embed objects inside of your report. Some examples are embedding Word documents or Excel spreadsheets. This is a feature that was brought forward from previous versions of Crystal Reports it isn't used much anymore.

The Hyperlink Object

The Hyperlink object is used for giving your report the ability to show a hyperlink to an on-demand subreport. When a user clicks on the hyperlink, the subreport opens in the viewer and gets its own tab. This creates an on-demand subreport. The user can toggle back and forth between the two tabs to see both reports.

The Special Field Object

The Special Field object is used for printing all report-related information. For example, it can print the current page number or the total number of pages. This field was created as a catch-all for all the miscellaneous types of report information that you need. Table 2-3 lists the available fields and what they mean. These fields are treated the same as a database field object. They can be added to the report by themselves, or included in a textbox object. They also have the same formatting options described in the Formatting Strings section.

Table 2-3. Special Fields.

Special Field	Description
DataDate	The date when the report was last refreshed.
DataTime	The time when the report was last refreshed.
FileAuthor	The file author that is stored with the report.
FileCreationDate	The date when the report was created.
Filename	The report's file name.
GroupNumber	The current group number.
GroupSelection	The group selection formula.
ModificationDate	The date the report was last modified.
ModificationTime	The time the report was last modified.
PageNofM	Prints "Page X of Y".
PageNumber	The current page number.
PrintDate	The date the report was printed.
PrintTime	The time the report was printed.
RecordNumber	The current record number.
RecordSelection	The record selection formula.
ReportComments	The report comments that are stored with the report.
ReportTitle	The report title.
TotalPageCount	The total number of pages.

3
Integrating Reports

After creating a report and formatting it to your satisfaction, the report needs to be integrated into the application. There are many ways to do this which makes chapter 3 one of the more complicated to understand, and yet one of the most important in the book. Understanding the best way to integrate reports into your application is essential for designing the best reporting solution.

There are a lot of different ways to integrate a report into an application, and some of the ways can be combined together. The different combinations create an interesting matrix of possibilities for you to choose from. If this is your first experience to writing reports with .NET, it may seem a little overwhelming. To try to make this as easy as possible to understand, I first give you an overview of each option and show its benefits and drawbacks. This lets you see the big picture and learn which options work for which types of applications. Afterwards, I break down each option and show the steps and programming code to integrate it into your application. This is done for Windows applications as well as ASP.NET applications. If you get to the examples and something doesn't make sense, just refer back to the beginning of the chapter for an explanation.

Sound good? Now let's get started!

The first reporting related decision is whether the report will be previewed before it gets printed. Many users like to see the report prior to printing it so that they can verify that it will print the information they are looking for. This is especially true for reports that let the user choose how to filter and display the data. For other reports you may not want to give the user the option to preview the report. An example is a reporting application that runs a batch print job at a scheduled time during the night. Requiring user intervention would cause the program to hang indefinitely until someone arrives to push the right button. Another example is a group of standardized reports that get printed every month. It is quicker for the user to select the reports to be printed by clicking on checkboxes and printing them all at once. Since the format and filters never change, making the user preview each report is unnecessary.

Previewing reports can be done in a Windows application as well as an ASP.NET application by adding a CrystalReportViewer control to the form/web page. Using the viewer control gives you many benefits. The first is obviously that the user can see what is going to be printed prior to printing it. If it isn't exactly what they want, they can close the preview window and repeatedly modify the report parameters until the report delivers the necessary information. No waste and another tree gets to live. Another benefit is that the viewer control has numerous built-in reporting functions that save you a lot of work. Rather than writing the code to export reports in different formats, the viewer has a button on the toolbar that does it for you. In fact, the viewer is so useful that I would guess that most of the applications you write are going to use it.

Of course, the viewer isn't ideal for every situation. It requires user intervention for it to be useful. Another downfall of the viewer is that it can only view one report at a time. You can't pass multiple reports to it. Displaying multiple reports simultaneously requires using multiple viewers.

If you decide to send reports directly to the printer and not use the viewer control, you are going to have to write the programming code to implement the functionality that you need. This could include writing code for printing the report, selecting the page range, and exporting the report to other formats. That's a lot of work! Of course, Part II of this book shows you all the programming code to do this, but it's still your responsibility to implement it in the application and test it.

If you don't use the viewer control, there will be a certain amount of code you always have to write. This code serves as the foundation adding more reporting functionality later. Part II of this book builds upon this code for advanced runtime customization.

To start coding, declare and instantiate an object variable from the ReportDocument class. The ReportDocument class has all the properties and methods necessary for loading and printing a report.

Note

Although I didn't mention it before, if you use the viewer control to preview a report, the ReportDocument class is still used to load and print the report. The viewer control does all the work for you behind the scenes and you don't have to be aware of it.

```
Dim MyReport As New CrystalDecisions.CrystalReports.Engine.ReportDocument
MyReport.Load("C:\Report1.rpt")
MyReport.PrintToPrinter(1, False, 0, 0)
```

The first line of code declares an object variable MyReport and instantiates a new instance of the ReportDocument class. As I said earlier, this ReportDocument class is responsible for managing all report related functions.

The second line of code calls the Load() method to load an Untyped report into memory. The last line of code calls the PrintToPrinter() method to send the report to the printer.

Note

Once you get into advanced programming techniques, you'll learn that there are times when you will use the viewer control and still create a ReportDocument object variable. This is when you want to perform runtime customization. See Part II of the book for more information.

The PrintToPrinter() method needs to be explained in more detail because the parameters passed to this method aren't intuitive. The parameters of the PrintToPrinter() method are listed in Table 3-1.

Table 3-1. PrintToPrinter() parameters.

Parameter	Description
nCopies	The number of copies to print.
Collated	Set to True to collate the pages.
startPageN	The first page to print. Set to 0 to print all pages.
endPageN	The last page to print. Set to 0 to print all pages.

The first parameter, nCopies, sets how many copies of the report to print. If more than one copy of the report is being printed, the second property tells whether to collate the copies or not. Passing False tells it to print each full report prior to printing the next copy. If nCopies is set to 1, then the collated parameter is ignored. The last two parameters, startPageN and endPageN, set the page range. Pass it the first page to print and the last page to print. To print the entire report, pass 0 to both parameters.

When using the viewer control, you also have the option of using a ReportDocument component (see the example in Chapter 1). The ReportDocument component gives you a visual way of telling the viewer which report you want to print. If you are writing a Windows application, adding the ReportDocument component lets you preview a report without writing any code whatsoever. The ReportDocument component also gives you the benefit of being in design mode and letting you see the name of the report that is going to be printed without having to read the programming code.

At this point in the chapter, you've had the chance to examine when you should use the viewer or when to send reports directly to the printer. If you use the viewer control, you have the option to add a ReportDocument component. The next major decision to make is how you want to instantiate the report object.

.NET applications are fully object oriented and thus everything is considered an object. Reports are no different. Crystal Reports .NET uses the ReportDocument class to manage all report functionality. Before a report can be printed, you must instantiate a new ReportDocument object variable and load the report into it. The steps for instantiating the object and loading the report are dependent on the report type and classification. A report can be classified as either Strongly-Typed or Untyped.

Strongly-Typed

When you create a report, .NET automatically creates a report class within the current project. Just like all the other classes in a project, the report is instantiated by its class name. Reports that are referenced by their class name are called Strongly-Typed reports. These reports are compiled into the applications executable file.

Untyped

When you save a .NET project, each report in the project is saved as a separate ".rpt" file. Although these files are saved in the same folder in your .NET application, they are independent of the project. In other words, you can copy these .rpt files to another folder and use them in another project. You could also copy them to a reporting library to be shared among many applications. Report files are separate from the current project have to be referenced by their filename and path. Reports that are referenced by their filename, rather than their class name, are called Untyped Reports. When you build your application, these reports will not be part of the application.

Each report type has its own benefits and drawbacks. Untyped reports are useful for applications that let the user choose from a dynamic list of reports. An example of this is when the user browses to a directory location and selects which report to print. You can also use Untyped reports to share reports between different applications. This lets you create a reporting library that is shared on the network. Another benefit is that if you distribute an application and later find that you have to modify a report, you can change the report without rebuilding the application. A drawback of using Untyped reports is that when deploying your application you have to remember to include each report file in the deployment project (unless its stored on the network). If you are writing an ASP.NET application, you also have to make sure that the security settings let you access the report.

Strongly-Typed reports are the easiest to work with in your application. Since the class is already part of your project, it is pretty simple to reference it by name. You also get the benefit of having the report compiled within the application's executable and you don't have to distribute the report file. There are two limitations however to using Strongly-Typed reports. First is that you can't reference reports designed by other applications or by the Crystal Reports stand-alone application. Second is that if you modify a report, you have to recompile and redistribute the application's executable.

Note

Report types are not mutually exclusive within an application. For example, choosing to open a report as a Strongly-Typed report doesn't mean you can't open other reports as Untyped reports. The purpose of classifying reports into types is for determining the best way to write your application. Each report can be opened in the way that is best suited for it.

That's it! Now that you've read about the various options available for integrating reports into your application, look at table 3-2 for a summary of each option.

Table 3-2. Summary of report integration options.

Integration Option	Key Points
CrystalReportViewer	Lets the user visually verify. Saves you programming effort.
Printing to the printer	Print report without user intervention. You have to instantiate a ReportDocument object variable. Could require a lot of programming. Useful for printing batch reports or printing standard reports.
Adding the ReportDocument component	Must be used with the viewer control. Gives you a visual way of specifying which report to print.
Untyped reports	Used when you have to specify the report by its filename. Allows you to share reports between applications. Report changes don't require recompiling and redistributing the application. Can be added to reporting library.
Strongly-Typed reports	Used when the report was created within the same application that is using it. Coding is easier than using Untyped reports because the report is a class within the project. Modifying a report requires recompiling and redistributing the application.

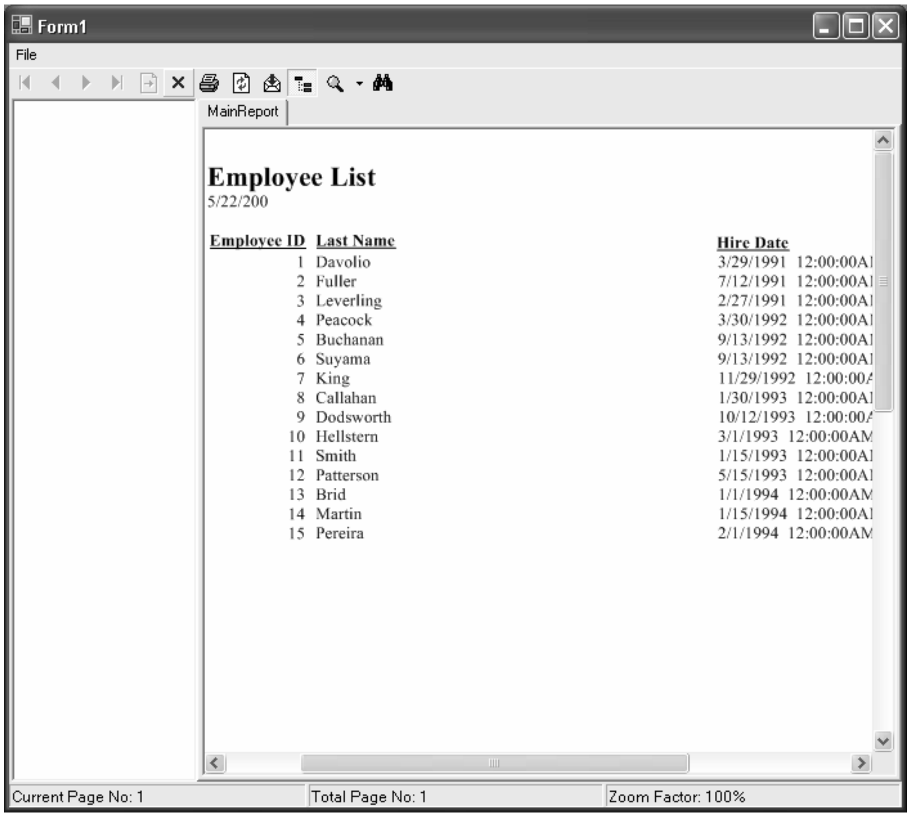
The next two sections take each option and show you the exact steps for implementing each one in your application. The first section is specific to Windows applications and the second section is for writing ASP.NET applications.

Windows Development

Each of the integration options presented in the previous section have a specific way that they must be implemented to work properly. This section discusses the details of each option and walks you through the steps to implement them in a Windows application.

Previewing Reports with the CrystalReportViewer

Most applications give the user the ability to preview a report before printing it. With Crystal Reports, this is controlled by the CrystalReportViewer control. In fact, using the viewer is the only way to preview a report in your application. The viewer is found in the Toolbox at the bottom of the other components. Double-click on it to add it to your form.



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

Figure 3-1. The CrystalReportViewer previewing a report.

Figure 3-1 shows the viewer previewing a report. Along the top of the viewer is a toolbar with a variety of navigational buttons. In the center is the report preview window with a Group Tree window to the left of it. Along the bottom is a status bar that shows the page number information and the current zoom factor.

By default, the toolbar buttons and Group Tree window are all enabled. Each of these features has a corresponding property that can be set in design mode and during runtime so that can turn them on or off. For example, a report that doesn't have any groups certainly doesn't need to show the Grouping Tree window. You may also want to turn off all these features so that you can create a customized preview form using your own buttons. Having a customized preview form lets you control the user interface by using your own style of buttons. This insures that your application has a consistent look and feel across all forms.

Tip

The status bar can't be modified. There are no properties to turn it on or off. Nor are there any properties to control what it displays. You can work around this limitation by extending the bottom of the viewer below the edge of the form or panel it is located in. This results in hiding the status bar.

Table 3-3 lists the properties of the CrystalReportViewer. The properties let you enable or disable the features of the viewer. Table 3-4 lists its methods. The methods let you implement the viewer's functionality by calling them from your own buttons. Most of these are self-explanatory, but the tables make good reference material.

Table 3-3. CrystalReportViewer Properties

Property	Description
DisplayBackgroundEdge	A background edge creates a border around the edge of the report page in preview mode. Setting this to False makes the edge of the page flush against the viewer's window.
DisplayGroupTree	Toggles the Group Tree window on and off.
DisplayToolbar	Toggles the toolbar on and off.
EnableDrillDown	Sets whether the user can drill down on reports.
ShowCloseButton	Sets whether the Close button is available.
ShowExportButton	Sets whether the Export button is available.
ShowGotoPageButton	Sets whether the GotoPage button is available.
ShowGroupTreeButton	Sets whether the GroupTree button is available.
ShowPrintButton	Sets whether the Print button is available.
ShowRefreshButton	Sets whether the Refresh button is available.
ShowTextSearchButton	Sets whether the Search button is available.
ShowZoomButton	Sets whether the Zoom button is available.

Table 3-4 CrystalReportViewer Methods

Method	Description
CloseView()	Pass a null value to close the current view. Pass a view name to close a specific view.
DrillDownOnGroup()	Drill down on a specific group. See Chapter 12 for details on implementing this method.
ExportReport()	Show the Export dialog box.
PrintReport()	Show the Print dialog box.
RefreshReport()	Refresh the report view. The user will be prompted for the parameters and logon information again.
SearchForText()	Pass a string to search for. If found, it returns True and moves to the page that has the string.
ShowFirstPage()	Move to the first page of the report.
ShowGroupTree()	Shows the Group Tree window. It doesn't take any parameters. There is no corresponding method to hide it.
ShowLastPage()	Move to the last page of the report.
ShowNextPage()	Move to the next page of the report.
ShowNthPage()	Pass an integer to move to that page number.
ShowPreviousPage()	Move to the previous page of the report.
Zoom()	Pass an integer to set the zoom level.

Binding Reports to the Viewer

The viewer needs to be told which report to display. This is called binding the report to the viewer. The property used for binding is the ReportSource property. There are three ways to set the ReportSource property:

using the ReportDocument component, passing it an Untyped report filename, and passing it a Strongly-Type report class name. Each method is covered in the next three sections.

Using the ReportDocument Component on the Viewer

Adding a ReportDocument component to the form is the easiest way to specify which report to print. Everything about it is visual and there is no code to write. Add it to your form by going to the Components section of the Toolbox and double-clicking on it. It will automatically display the Choose a ReportDocument dialog box. This lets you select which report to display. The dropdown control lists all the reports that are part of your project. Once you select which report to use, the dialog box closes and it the component gets added to your form.

Note

The ReportDocument component has to be on the same form as the viewer control. The viewer can't reference a ReportDocument component on another form.

After the component is added to the form, tell the viewer to use it by clicking on the viewer's ReportSource property and selecting the ReportDocument's name from the dropdown list.

When you run the application and open the form, the viewer automatically displays the report to the user. You don't have to write any code.

Using Untyped Reports with the Viewer

Untyped reports are, by definition, referenced by their filename. The viewer loads the report into memory and displays it to the user. The viewer's ReportSource property has to be passed the report's filename.

Setting the ReportSource property can be done anywhere within your form. For example, you could call it in response to the user clicking on a button or a menu item. In this example (and almost all the examples in this book), I put the code in the form's Load() event. I do this because it is easy to understand and it makes the form preview the report immediately. In this example the report used is the Employee List report created in Chapter 1. You should replace the report filename with your own report.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles yBase.Load
    CrystalReportViewer1.ReportSource = "Employee List.rpt"
End Sub
```

Using Stongly-Typed Reports with the Viewer

Strongly-Typed reports are referenced by their class name. Create a new instance of the report class and pass it to the viewer's ReportSource property. As mentioned in the last section, the code in this example is called within the form's Load() event so that the report is displayed immediately upon opening the form. The Employee_List class name is from the example report in Chapter 1. Replace it with the class name of the report in your own application.

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

Notice that the report's class name is "Employee_List" and the actual report name is "Employee List". When there are spaces in the report name, the class name that .NET creates will use underscores instead of spaces.

Printing Reports Directly to the Printer

If you decide not to user the viewer control for previewing reports, you will have to write all the code for loading and printing reports. The following sections show you examples of how to instantiate report objects and send their output to the printer.

At this point in the book, the amount of code you need to write is still very limited and it is easy to understand. Once you get into Part II of this book, you'll see that this code serves as the foundation for performing advanced runtime customization of your reports.

Printing Untyped Reports Directly

Printing Untyped reports without using the viewer requires a bit more coding than the examples you've seen so far. First you must declare and instantiate a ReportDocument object. The ReportDocument object has methods for loading the report into memory and printing it. Call the Load() method and pass it the report's filename. Call PrintToPrinter() to send the report to the printer.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
  Dim MyReport As New CrystalDecisions.CrystalReports.Engine.ReportDocument
  MyReport.Load("Employee List.rpt")
  MyReport.PrintToPrinter(1, False, 0, 0)
End Sub
```

This example loads the Employee List report in the MyReport object variable. Then it calls the PrintToPrinter() method to print a single copy of the report, including all pages.

Printing Strongly-Typed Reports Directly

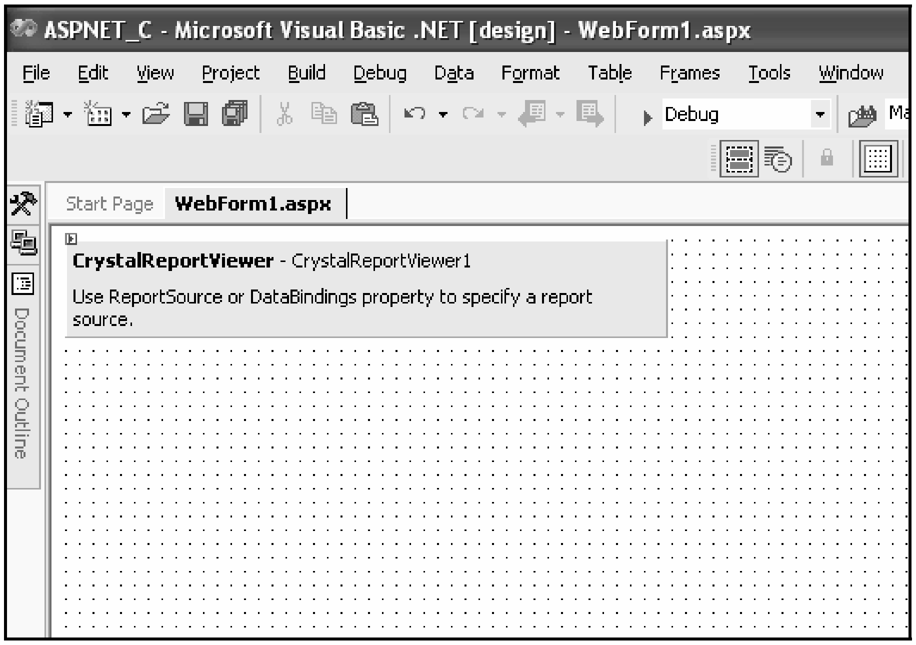
Printing Strongly-Typed reports is a little easier than printing Untyped reports. With Strongly-Typed reports, the report's class name is part of the project and can be instantiate directly. Unlike Untyped reports, you don't have to use the generic ReportDocument class and you don't have to worry about loading the report from a file. After instantiating the report object, call the PrintToPrinter() method to send it to the printer.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
  Dim MyReport As New Employee_List
  MyReport.PrintToPrinter(1, False, 0, 0)
End Sub
```

This declares the object variable MyReport to be of type Employee_List and instantiates it. For your application, use the class name of the report that you want to print. The PrintToPrinter() method prints a single copy of the report.

ASP.NET Development

The CrystalReportViewer control is the interface for previewing reports in an ASP.NET application. Adding a viewer control to a web page is as simple as dragging the control from the Toolbox onto the web page. After it is added to an ASP.NET page, it is represented by a simple rectangle.


ASPNET_C - Microsoft Visual Basic .NET [design] - WebForm1.aspx
File Edit View Project Build Debug Data Format Table Frames Tools Window
Debug
Start Page
WebForm1.aspx
Document Outline

CrystalReportViewer - CrystalReportViewer1

Use ReportSource or DataBindings property to specify a report source.

Figure 3-2. The viewer control as positioned on an ASP.NET page.

If you worked through the Windows example where you added the viewer control to a form, you will immediately notice that the ASP.NET viewer looks different than the Windows viewer. When you add the viewer to an ASP.NET page, it only displays a simple rectangle. It doesn't show you a template of what the viewer looks like.

Although the viewer control is very small when you add it to the web page, when you run the application it automatically grows to fill the remainder of the web page. It expands horizontally across the page as well as vertically down the page. This ensures that the report is readable when the page is displayed.

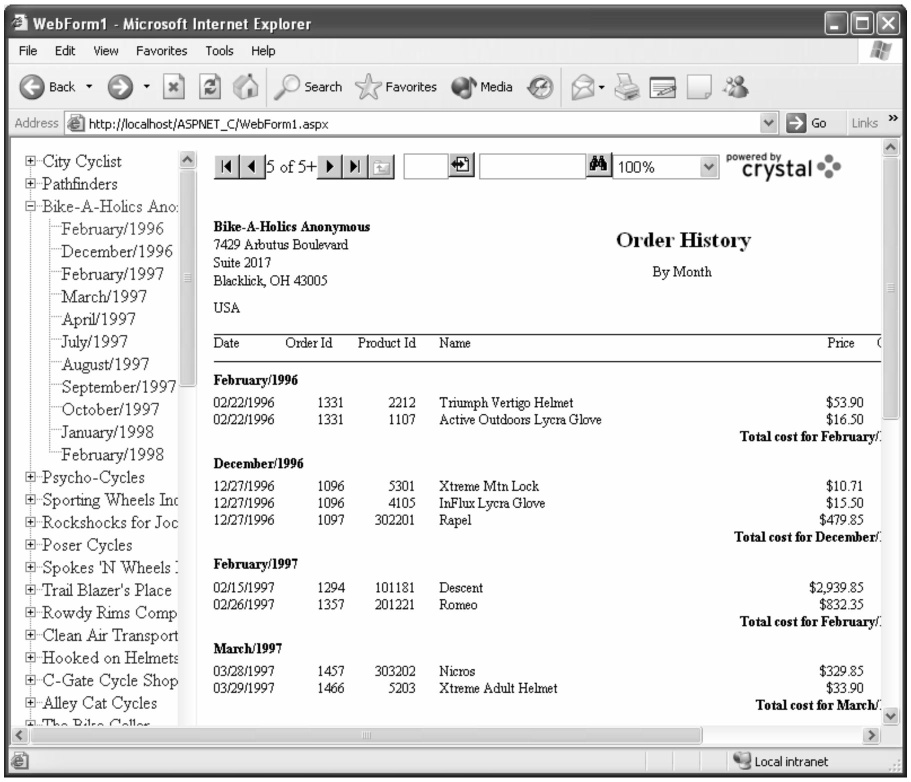
The viewer has a BestFitPage property (default value of True) that controls whether it fills the page or not. If you set this property to False, the viewer control will not resize itself and the report will be displayed within the bounds of the viewer. Setting it to False is useful when you want to limit the amount of space on the web page that is allocated to viewing reports. Obviously, if you decide to set BestFitPage to False, you should resize the viewer to make it large enough to comfortably display your report.

When BestFitPage property is True, the control does not expand upward. It only expands downward and across. Any information above the viewer control remains intact and properly formatted.

Note

If the Page Layout is set to FlowLayout, all information below the viewer will be pushed down the page so that it appears after the viewer control. If the Page Layout is set to GridLayout, the information after the viewer control will not be pushed down. The ASP.NET controls will overlap the viewer control. This could either make for a very messy report or you could use it to add some interesting formatting effects.

As you can see in Figure 3-3, when the viewer is displayed in an ASP.NET page, it looks very similar to the report preview that you've seen with Windows applications.


WebForm1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address http://localhost/ASPNET_C/WebForm1.aspx
Go
Links
5 of 5+
100%
powered by crystal
Local intranet

City Cyclist
Pathfinders
Bike-A-Holics Ano
February/1996
December/1996
February/1997
March/1997
April/1997
July/1997
August/1997
September/1997
October/1997
January/1998
February/1998
Psycho-Cycles
Sporting Wheels Inc
Rockshocks for Joc
Poser Cycles
Spokes 'N Wheels
Trail Blazer's Place
Rowdy Rims Comp
Clean Air Transport
Hooked on Helmets
C-Gate Cycle Shop
Alley Cat Cycles

Bike-A-Holics Anonymous
7429 Arbutus Boulevard
Suite 2017
Blacklick, OH 43005
USA

Order History
By Month

```
Date        Order Id  Product Id  Name                          Price

February/1996
02/22/1996  1331        2212      Triumph Vertigo Helmet       $53.90
02/22/1996  1331        1107      Active Outdoors Lycra Glove  $16.50
                                            Total cost for February/

December/1996
12/27/1996  1096        5301      Xtreme Mtn Lock              $10.71
12/27/1996  1096        4105      InFlux Lycra Glove           $15.50
12/27/1996  1097      302201      Rapel                       $479.85
                                            Total cost for December/

February/1997
02/15/1997  1294      101181      Descent                   $2,939.85
02/26/1997  1357      201221      Romeo                       $832.35
                                            Total cost for February/

March/1997
03/28/1997  1457      303202      Nicros                      $329.85
03/29/1997  1466        5203      Xtreme Adult Helmet          $33.90
                                            Total cost for March/
```

Figure 3-3. The report preview in an Internet Explorer browser.

The ASP.NET viewer control has some interesting differences between it and the Windows viewer.

There are no tabs shown along the top of the ASP.NET viewer. When you drill down on reports in a web viewer, it opens a new page rather than showing all the tabs on a single page. There is also no Print button. A very disappointing aspect of the viewer is that it can't send a report to the printer. You have to use the browser's built-in print functionality to print the web page. This has the adverse effect of printing non-professional reports because it prints everything in the browser window. This includes the toolbar, the group tree and the web page information. Printing professional reports requires advanced programming techniques which are discussed in Chapter 19.

The last thing you might notice is the "Powered by Crystal" icon at the top of the toolbar. Although this is totally harmless, many people don't like appearing it on their reports.[4] You can get rid of this logo, or even replace it with your own, by deleting or overwriting the GIF file on your computer. You can find it within the Visual Studio .NET installation folder under "..\crystal reports\viewers\images\toolbar\logo.gif"

The properties of the Web viewer control are similar to that of the Windows viewer control. It has properties for changing the look of the viewer by hiding the toolbar, hiding the toolbar buttons and hiding the Group Tree window.

Although the Windows and Web viewer controls have properties that perform the same functionality, they often have different names. There are also certain properties that are unique to each control. Table 3-5 compares the common properties of each control so you can see the similarities and differences.

[4] Later in the book you'll see more examples of where Crystal Decisions has decided to stamp their logo within your report output. I'm sure their marketing department loves this, but I consider it unprofessional. With a little work all these logos can be removed.

Table 3-5. Comparing the property names of the two viewer controls.

Windows Viewer	Web Viewer
DisplayGroupTree	DisplayGroupTree
N/A	DisplayPage
DisplayToolbar	DisplayToolbar
Dock	BestFitPage
EnableDrillDown	EnableDrillDown
N/A	DrilldownTarget
N/A	HasDrillUpButton
ReportSource	(DataBindings) | ReportSource
SelectionFormula	SelectionFormula
ShowCloseButton	N/A
ShowExportButton	N/A
ShowGotoPageButton	HasGotoPageButton
ShowGroupTreeButton	N/A
ShowPageNavigationButtons	HasPageNavigationButtons
ShowPrintButton	N/A
ShowRefreshButton	HasRefreshButton
ShowTextSearchButton	HasSearchButton
ShowZoomButton	HasZoomFactorList
N/A	HyperlinkTarget
N/A	PageToTreeRatio
N/A	SeparatePages

Both controls have properties for hiding the different buttons, but they use different names for them. For example, the properties of the Windows viewer that hides the toolbar buttons are prefixed with "Show", but the web viewer properties are prefixed with "Has". The Dock property of the Windows viewer is similar to the BestFitPage property in the Web viewer.

There are certain properties that only belong to the Windows viewer and not the web viewer. For example, there are more buttons in the Windows toolbar than the web toolbar. As a result, the properties to hide these buttons aren't in the web viewer. The web toolbar is missing the buttons for Close, Export, GroupTree, and Print. The Close button is missing because there are no tabs displayed for drilling down into the report detail. The Export and Print button are missing because the web viewer can't perform either function. The GroupTree button is missing because the user isn't allowed to directly turn this off. If you want to give the user this ability, add a button to the web page that toggles the value of the DisplayGroupTree property.

The web viewer also has properties that aren't in the Windows viewer. The DisplayPage property hides the report preview when it is set to False. The web viewer also has properties that are HTML target values. When initially opening the report's web page or opening a sub-report, you can set whether the new page opens in the same browser window or in a new window. Table 3-6 shows the possible values. These values can be assigned to the HyperlinkTarget and DrilldownTarget properties.

Table 3-6. Hyperlink target values.[5]

Target String Value	Description
_blank	Opens the page in a new, unframed window.
_parent	Opens the page in the immediate frameset parent.
_self	Opens the page in the current frame.
_top	Opens the page in a full, unframed window.

The Windows viewer uses tabs along the top of the form to display different pages of data as the user drills down into the report for more information. As mentioned earlier, there are no tabs in the web viewer control. When a user drills down into group details and sub-reports, a new web page is generated to display the information. Rather than using tabs to look at the different pages, the user has to click on the Back and Forward buttons. To make this a little easier to navigate, the web viewer control has a button for moving up to the parent level. This is called the DrillUp button. When report is at the top-most level, this button is disabled.

The PageToTreeRatio property sets how wide the Group Tree control is. It represents the relationship between how wide the viewer is compared to how wide the Group Tree is. The number entered for this property tells how many units the report page is compared to one unit of the Group Tree control. For example, if the number is 1, then the two areas have a 1:1 ratio and they are equal sizes. Thus, the page is split in half. If the number is 4, the two areas will have a 4:1 ratio, which makes the width of the Group Tree control use 20% of the total browser width. A good number to start with is 4 or 5.

The SeparatePages property determines whether the report uses different pages or not. If the property is set to True, each web page displays a single report page. If the property is False, the entire report is displayed on a single web page and the user has to scroll up and down.

Caution

Setting the SeparatePages property to True is the most efficient because Crystal Reports only has to process enough information to display one page at a time. Setting the SeparatePages property to False can cause a long delay to rendering the page. Crystal Reports has to process the entire report before it can display it.

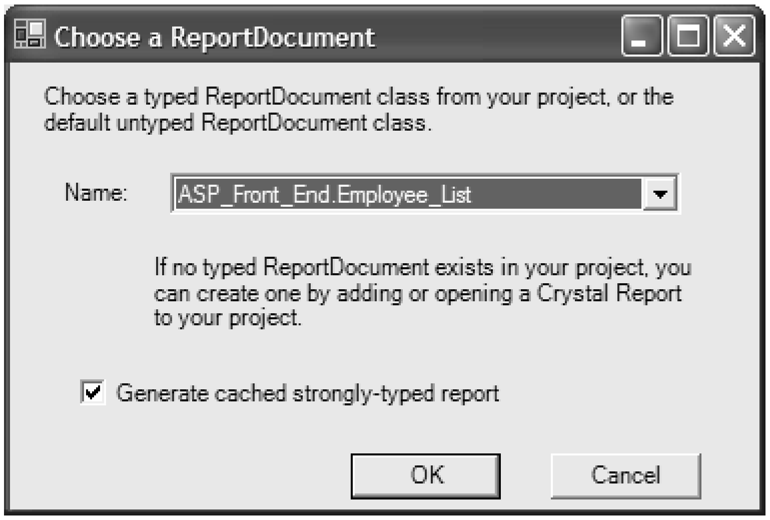
Binding Reports to the Web Viewer

After adding a viewer to the web page and setting its properties, the report needs to be bound to the viewer. This can be done with any of the three options listed earlier in the chapter: using the ReportDocument component, using Untyped reports, and using Strongly-Typed reports.

Using the ReportDocument Component on the Viewer

The ReportDocument component makes it easy to specify which report the viewer is supposed to display. 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. 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.

[5] These are the standard target values used with HTML hyperlinks.


Choose a ReportDocument

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

Name: ASP_Front_End.Employee_List

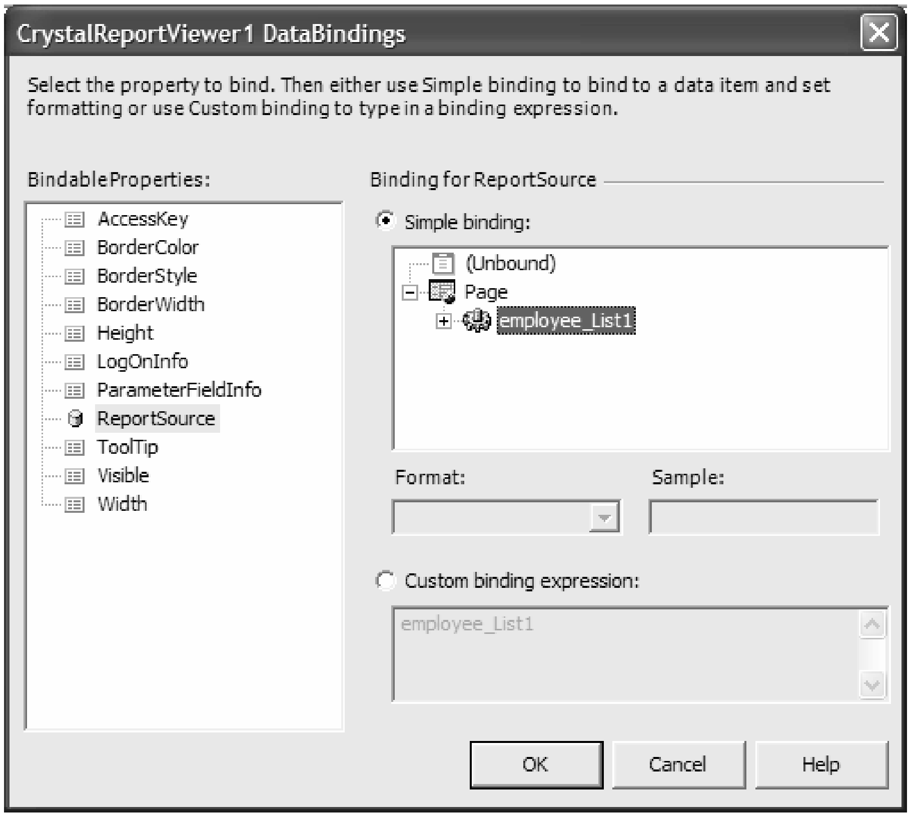
If no typed ReportDocument exists in your project, you can create one by adding or opening a Crystal Report to your project.

[✓] Generate cached strongly-typed report

Figure 3-4. 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

Figure 3-5. The DataBindings dialog box.

Click on the ReportSource property listed on the left. Then expand the Page item in the Simple Binding window. It shows you the reports that were added to the page using ReportDocument component.

Select the report name and click the OK button. Once the dialog box closes, a preview of the report is immediately displayed. Unlike Windows development, the viewer component shows you what the report looks like while you are in design mode.

When the application runs, you have to tell the viewer to use the DataBindings property. This is done by calling the DataBind() method. Put this code 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
```

Run the web application and you'll see the Employee List report automatically previewed when the page opens.

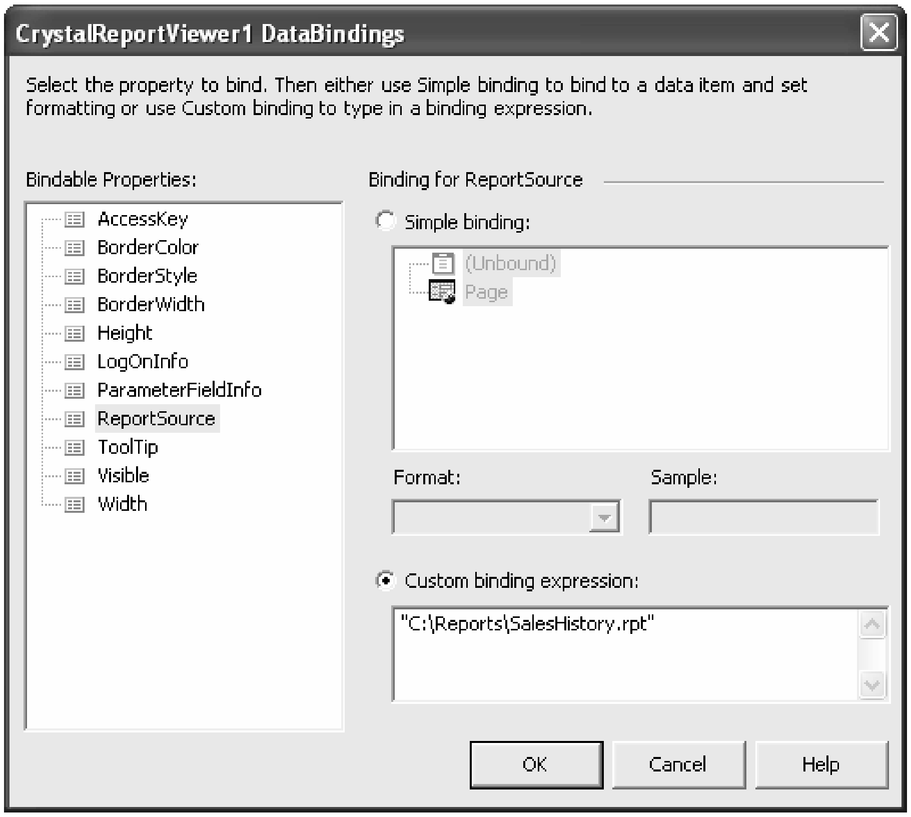
Using Untyped Reports with the Viewer

Untyped reports are referenced by their filename and they are external to the application's executable code. Open an Untyped report by passing the filename to the ReportSource property of the viewer. In this example, I set the ReportSource property in the Load() event of the page. I also pass it the name of the Employee List report that I created in Chapter 1. For your application you should pass the filename of the report you want to display.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    CrystalReportViewer1.ReportSource = Server.MapPath("Employee List.rpt")
End Sub
```

A second way to bind an Untyped report to the viewer is to set the ReportSource property in the DataBindings dialog box. Look at the Property Window for the viewer 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.

When the dialog box opens, click on the ReportSource property and then click on the Custom Bindings Expression option. Within the text area enter the report's full path name and enclose it in quotes. You should also be aware that there is no Browse button for this property. You have to type in the file path exactly.


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

Format:

Sample:

Custom binding expression:

"C:\Reports\SalesHistory.rpt"

Figure 3-6. Setting the custom binding expression.

Caution

Be careful when entering the report filename in this dialog box. It is a common source of problems. First of all, the file path must be accessible by the report server and you must have permissions. Secondly, always put quotes around the file path string. Otherwise you will get an error when the viewer attempts to open the report.

Using Strongly-Typed Reports with the Viewer

Strongly-Typed reports are referenced by passing an instance of the report class to ReportSource property of the viewer. In this example I pass a new instance of the Employee_List class to the ReportSource property. For your application you would pass the class name of the report you want to display.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles MyBase.Load
    CrystalReportViewer1.ReportSource = New Employee_List
End Sub
```

Caution

When using Untyped or Strongly-Typed reports, don't call the viewer's DataBind() method. If you call DataBind() after setting the ReportSource property, the ReportSource value is reset and the report won't display. I see this mistake a lot because people assume that calling the DataBind() method is a requirement for using the viewer. This isn't the case at all. It's only necessary if you set properties within the DataBindings dialog box. If you do need to call the DataBind() method, call it prior to setting the ReportSource property in code

5

Using Parameters and Formulas

As Crystal Reports has become increasingly advanced over the years, the number of ways to customize reports has also increased. However, passing data to a report hasn't changed much. Parameters are still the preferred method of passing data to a report. This chapter explores how to create parameters, query the user for input, and use parameters to customize a report. Parameters have been a very effective way to getting user input, but parameters are showing their age and are ripe for improvement now that Crystal Reports have been integrated into .NET. Formulas address some of the parameters' weaknesses. They are easier to work with during runtime and are less complex to set up. This is discussed in more detail in Chapter 16.

Inputting Parameters

Crystal Reports considers parameters to be the programming language equivalent of a constant data type. The parameter is assigned a value when the report loads and that value never changes. A parameter is like any other field on the report: It can be displayed on the report, used in filters, and used to change the formatting of report objects.

In their simplest form, parameters are used as an easy way to let the user enter a value. At another level, they can be thought of as a way of creating advanced input boxes. Parameters can have default values defined in such a way that the user is able to enter the value by selecting it from a predefined list in the combobox. The programmer controls how many options a user has in the combobox. The section on default values details the setting up of parameters in this way. When a report loads, parameters get their values by displaying a dialog box to the user. This dialog box prompts the user for information and it has input fields for the user to enter one or more values. Once the user closes the dialog box, the report uses the user input to generate the report. An example of this is a report that prints records between a valid data range. Parameters confine the beginning and ending date ranges.

Adding Parameters

Parameters are another type of report object. The steps to add and modify them are similar to what you've already been doing. Add a parameter by clicking on the Field Explorer tab (on the left side of the IDE by default) and right-clicking on Parameter Fields. Then, select the New menu option. Clicking on an existing item allows the option to edit, delete or rename the item. Figure 5-1 shows this menu.

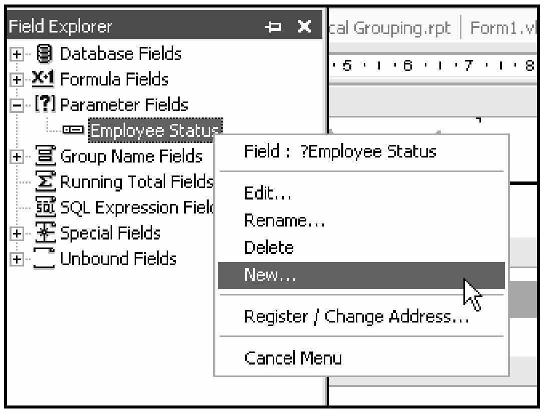


Figure 5-1. Menu option for adding a new parameter.

Once you select New, the Create Parameter Field dialog box appears, shown in Figure 5-2.

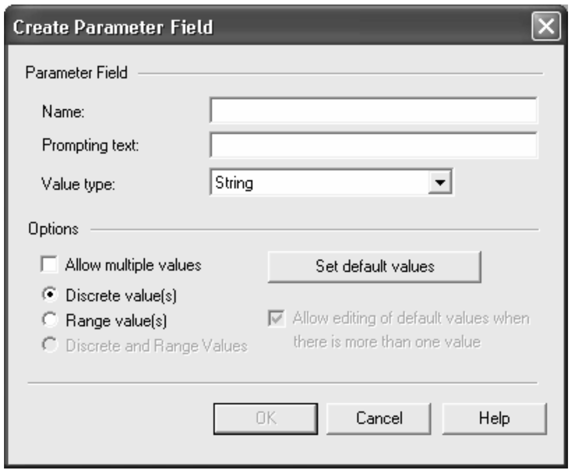


Figure 5-2. The Create Parameter Field dialog box.

This dialog box has three textboxes at the top, for entering the necessary properties: the parameter's name, the prompting text and the value type. The Name property is how the parameter is referenced in the report. Running the report triggers the display of the Prompting Text, which should describe what the user is asked to enter. The Value Type property selects the data type for the parameter field. All the available data types are listed in this dropdown box.

The lower half of the dialog box lets you set the options for the data that the parameter can store. Table 5-1 describes four options.

Table 5-1. Options for parameter fields

Option	Description
Discrete value(s)	The user must enter must a single value.
Range value(s)	The user enters two values that are the beginning and ending points of a range. The range includes the values entered. For example, if you entered a range of 1,000 and 1,999 then it would include all numbers from 1,000 up to and including 1,999.
Discrete and Range values	The user can enter both discrete and range values.
Allow multiple values	Allow a parameter to accept more than one value.

The option to allow multiple values is used with discrete values and range values. For discrete values, the user can enter multiple single values and they are each treated individually. For range values, the user can enter multiple sets of ranges and each range is treated separately from the other ranges entered. A parameter can also have a collection of both discrete values and range values.

Boolean parameters can be thought of as being similar to checkboxes or option buttons. With checkboxes, each value is independent of the other. That is, selecting or changing one checkbox has no effect on other checkboxes. This is the default behavior of Boolean parameters. Option buttons are used in groups and each is mutually exclusive of the other. Selecting one will automatically turn off all others in the same group. The Place in Parameter Group option lets you place a Boolean parameter in a parameter group. Give it a group number and set whether the parameters in that group are mutually exclusive to each other. If the parameters are mutually exclusive, only one Boolean parameter can be assigned to True at a time. If the user sets two or more parameters to True, only the most recent will keep its value; the others are reset to False.

The Create Parameter Field dialog box for Boolean data types has different options. When you change the Value Type property to Boolean, the lower half of the dialog box changes to reflect the new options (see Figure 5-3).

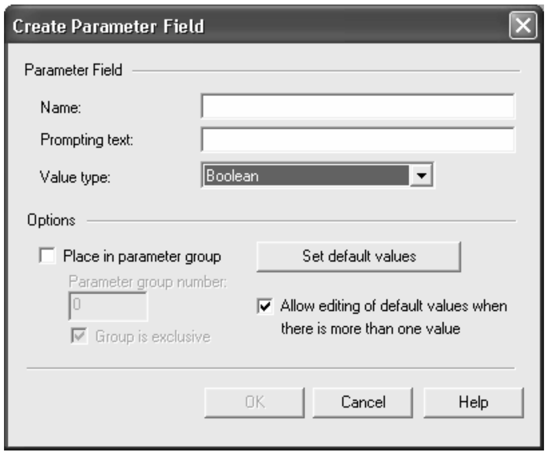


Figure 5-3. The Create Parameter Field dialog box for Boolean data.

Setting the Default Values

Default values give the user a list of values to choose from, which saves the user from the chore of memorizing all the available values that could be entered. Default values also restrict what can be entered. This prevents a user from making a typographical error.[6]

One drawback of using default values is that they create additional overhead and increase the risk of error because it goes against the methodology of Object Oriented Programming (OOP). The goal of writing applications using OOP techniques is to consolidate all the information about a business object within its respective classes. This gives you encapsulation and makes it easier to maintain your code. Setting default values within a report breaks this rule because you are storing information about an object outside of its classes. In fact, this data is being stored in a separate file entirely. This requires additional documentation stating that all rules regarding an object's default values must also be replicated to the related reports. If there is a bug in the report, both the report and the business objects would have to be debugged. This creates additional work during testing.

The best method is to not set default values within the report; instead use the .NET application to maintain this. Since the .NET front end is used to get input from the user before printing the report, logic can be added here to pre-populate the front end with the appropriate default values that are derived from the class.

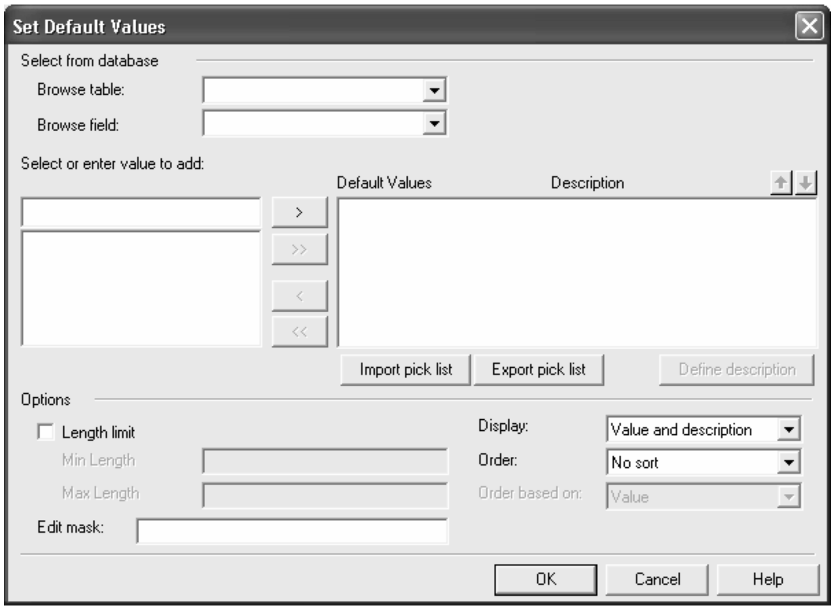
Of course, not everyone is going to write fully compliant OOP applications, nor will every report warrant the extra time required to fully integrate it into the .NET application. That being the case, this chapter explains how to create and use default parameters within a report.

Note

Default values aren't used with ASP.NET applications. With an ASP.NET application, Crystal Reports isn't able to prompt the user to enter a value for each parameter. You are required to set each parameter during runtime prior to viewing the report. Thus, default values lose their significance with ASP.NET applications.

[6] In CR.NET default values have numerous bugs and do not work as expected. Make sure you have the latest service pack if you plan on using default values in your reports.

Default values are added by clicking on the Set Default Values button on the Create Parameter dialog box. This brings up the Set Default Values dialog box. As Figure 5-4 shows, this is a fairly complex dialog box. This is because it allows the entry of every possible combination of values for each parameter type. That's a lot of combinations!



Display: Value and description
Order: No sort
Order based on: Value

Figure 5-4. The Set Default Values dialog box for string data types.

It was mentioned earlier that default values can be designed so that the user is presented with a listbox showing all the possible values. The Set Default Values dialog box is where this is done. Within this dialog box, one can create either a single default value or a list of default values. If a single default value, the input box is pre-populated with that value.[7] If you create a list of default values, the user, when entering a value for the particular parameter, can select one of those values from a combobox.

The middle of the dialog box is the central part of setting default values. The left side is the place to enter a default value in the textbox. Click on the right arrow to move it to the list of possible default values. There is no limit to the number of values added to the right listbox.

An alternative to manually typing all the default values by hand, if entering a lot of default values, is to pull them from existing tables. This neat trick is performed by selecting from the top two dropdown boxes a table and a field to get the values from. Once the two items are selected, the values are pulled from the field and put into the listbox on the left. Clicking on the arrow keys allows one to select and transfer to the default value list any number of these items. Alternately, to import a list of default values from a text file, click on the Import Pick List button.

Tip

The options to pull a list of values from a table or from a text file are better served by creating your own interface using .NET. Both of these options are just a mediocre attempt at creating a data-bound combobox. In Crystal Reports, there is no way to create a data-bound control for parameter values. This is just too complex for a reporting tool to implement. So presenting a static list of values from a table or text file is the next best option. Using this technique results in the problem that if your database were updated, you would have to come back to this report and update the list of possible values. But

[7] This requires installing the latest service pack.

with .NET report integration, you can provide the user with a true data-bound combobox on a form in your application. This is a more practical solution to implement and it insures that the user always has a current list of values to choose from.

There are two options that are unique to string values. Refer again to Figure 5-4. You can limit the length of a string size and set the edit mask in the lower left corner of the Set Default Values dialog box. Limiting the length of the string allows you to set the minimum length and the maximum length. However, these options do not exist if using parameters with a numeric data type. Instead, set a valid range by entering minimum and maximum values.

The bottom right corner of the dialog box is the place to go to for setting a description for each value, as well as the sorting order. This is similar to the functionality of a data-bound combobox because of the option to show a description rather the actual value.[8] This is useful for having the user select a table's primary key by clicking on the description rather than the numeric id. The values can be sorted in ascending, descending and natural order.

Closing the Set Default Value dialog box brings back the Create Parameter Field dialog box. Now that the parameter has a stored default value, the checkbox Allow Editing of Default Values is no longer grayed out (it is checked by default). When this is checked, the user can enter a new value in addition to selecting an item in the list of default values. Unchecking this option then requires the user to select an item from the list of values, which prevents him or her from entering an invalid value.

Entering Parameters when Running Reports

When a report has parameters, the user is prompted to enter values when the report is loaded, but before it is printed. After the user enters the values, the report is shown in the viewer or sent to the printer. Figure 5-5 is the Enter Parameter Values dialog box.

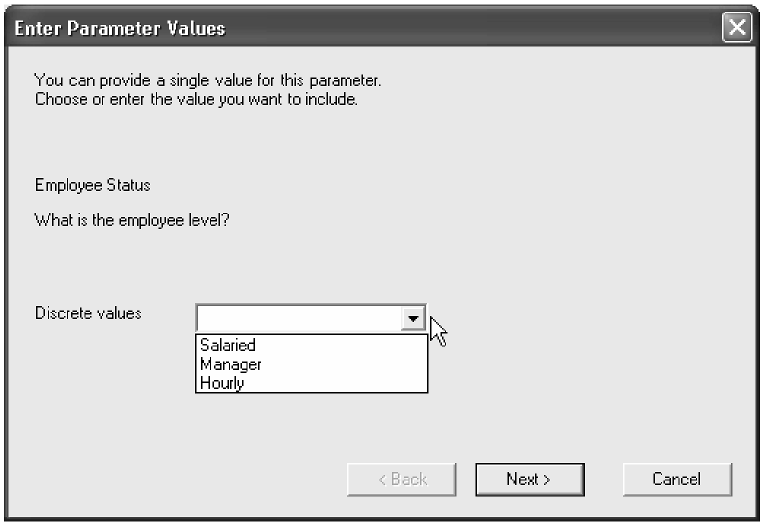

Enter Parameter Values
You can provide a single value for this parameter.
Choose or enter the value you want to include.
Employee Status
What is the employee level?
Discrete values
Salaried
Manager
Hourly
< Back
Next >
Cancel

Figure 5-5. The Enter Parameter Values dialog box.

At the top is a description of what information can be entered. Below that is the prompt that is assigned to the parameter. The bottom of the dialog box is where the user enters the value(s) for the parameter field. Click on the Next button to go to the next field. Clicking on the Cancel button at any time cancels the report. Nor will it be displayed in the viewer. If the user refreshes the report, this dialog box gets displayed again.

[8] This requires installing the latest service pack.

If you have used Crystal Reports 8.5, then you might notice that .NET treats mutually exclusive Boolean parameters differently than 8.5 does. With 8.5, mutually exclusive Boolean parameters were grouped together as a single step on the dialog box and you selected which one should be set to `True`. The other parameters within that group remained False. With .NET, this is no longer the case. Each parameter is shown separate from the others. If you select `True` for more than one parameter, only the most recent one will be `True`. The others will be reset back to False. Unfortunately, .NET doesn't tell you that this is happening nor does it give any indication that the parameter is part of a mutually exclusive group. Once again, this is just another reason to manage default values within your .NET application and not rely upon the report.

Caution

If you run a report and get the error "Operation illegal on linked parameter" it is because a formula references a parameter, but that parameter isn't used on the report. To fix this error make sure that the parameter appears on the report.

13

Data Connectivity

The backbone of every report is the data it prints. Large corporations merge data on servers into reports that consolidate and chart information from a dozen or more tables. Small businesses optimize their report distribution and expand their client base by providing their data in an XML format and letting companies from around the world generate reports on it. Home offices often do simple tasks such as tracking and printing monthly sales figures from an Access database or an Excel spreadsheet.

Crystal Reports is designed to work with many types of data. Reports can be generated regardless of where the data is stored; SQL Server, MS Access or even the Outlook email repository. Crystal Reports affords a many ways to connect to databases; learning each method can be quite an undertaking. This chapter, along with Chapter 17, sorts out these options and presents them in an easy to read format. You can determine which method best meets your needs and how to quickly implement it.

All database connectivity is built around one of two models: the Pull Model and the Push Model. The Pull Model is the simplest to implement and is very easy to learn because it doesn't require writing any programming code. Reports designed to use the Pull Model make everything automatic. Crystal Reports does all the dirty work: creates the connection, reads the data, populates the report and then closes the connection. The Pull Model is covered in this chapter. The Push Model, which is covered in Chapter 17, is just the opposite. You write the code to do all the work in your program. You have to open the connection, get the data into memory, pass the data to the report, and close the connection.

So why would anyone ever want to use the Push Model? Who would want to when they know that the Pull Model Crystal will do everything for them? The answer is no different from any other choice your make when writing software. Tasks that require more effort allow more functionality. Since the Pull Model is very simple, it's less flexible than the code intensive Push Model.

This chapter focuses on connecting to databases using the Visual Studio IDE. The IDE makes it easy to connect to a data source and generate reports without having to write any programming code. Chapter 17 in Part II of this book shows the how to's of solving more complex reporting problems by writing programming codes that connects to data sources using the ReportDocument object.

Note

If you are using datasets in a report, this is part of the Push Model. See Chapter 17 for implementing datasets.

Implementing the Pull Model

The Report Expert uses the simplest form of the Pull Model to create reports. Within the Report Expert is the Database Expert, where the data source(s) that the report connects to and the tables that have the data are defined. The Report Expert, generates a default report layout and builds, behind the scenes, all the data connections necessary. The only thing you have to do is call the proper method to either preview or print the report. All the examples in the book prior to this chapter used the Pull Model. This was done so that you could focus on report design and layout issues, unencumbered with worries about the intricacies of database. As you can see, implementing the examples didn't require any knowledge about data connectivity.

However, there are three aspects to the Pull Model you need knowledge of. They are: adding a data source, linking tables, and using multiple data sources.

Adding Data Sources

The Pull Model uses the Database Expert dialog box as the interface for selecting data sources for your report. It is a visual expert for opening data sources, finding tables within the data sources, and linking their related fields together. To get to it, run the Report Expert, or right-click on an existing report in design mode and select Database | Add/Change Database.

The Database Expert uses two tabs: the Data tab and the Links tab. The Data tab, shown in Figure 13-1, is the way to add data sources to the report. It shows two windows: Available Data Sources and Selected Data Sources.

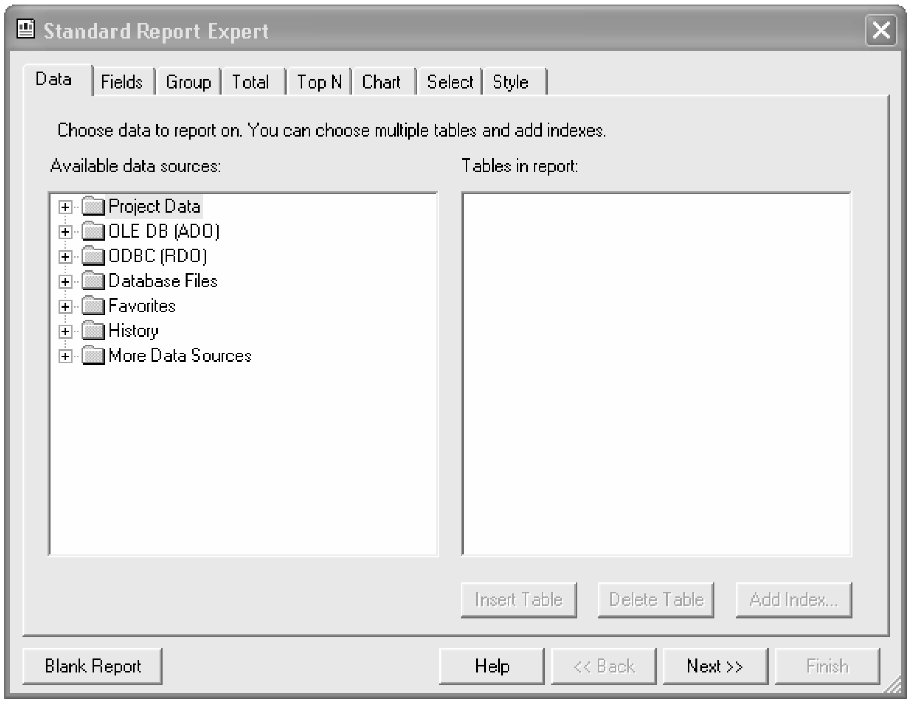


Figure 13-1. The Data Tab of the Database Expert.

The Available Data Sources window lists different categories of data sources. Table 14.1 lists these available data source categories.

Table 13-1. Available Data Sources

Data Source	Description
ADO.NET DataSets	Found within the Project Data category, these are the DataSet classes listed in the Visual Studio Project Explorer window. They give you the flexibility to tie your report to virtually any type of data.
Current Connections	Found within the Project Data category, select from any connections already established within the Server Explorer.
OLE DB (ADO)	Builds a connection string to access data sources using an OLE DB driver. It gives you a two screen wizard which asks you for the data source to connect to and any relevant information regarding its location and logon information.
ODBC (RDO)	Connect using an ODBC driver by selecting an existing System DSN[9].
Database Files	Select a PC database file using its file location.
Favorites	Choose from commonly used data sources you added to your Favorites list. Existing items can be added to the favorites list by right-clicking on them and selecting Add To Favorites. A Favorite can be renamed by clicking on it and pressing the F2 key.
History	Choose from data sources that have been used for other reports within the current project.
ADO.NET (XML)	Found within the More Data Sources category, it retrieves records by specifying an XML file path. You can also select an existing ADO.NET dataset.
Access/Excel (DAO)	Found within the More Data Sources category, it retrieves records using a DAO recordset that accesses a Microsoft Access database or Excel spreadsheet.
Field Definitions Only	Found within the More Data Sources category, it specifies a field definition file. Per the documentation, this is only for backwards compatibility and isn't used for new development.

To select the data source from the Database Expert dialog, click on the proper category node to expand it. This triggers a dialog box which asks for information about the data source. The dialog changes to match the needs of each data source. This information can range from a simple file path to a database server name and the appropriate logon credentials. Upon entering the information, the dialog box closes and the data source name is shown in the Database Expert and listed under its category. Under the data source name is the list of available tables, views and stored procedures.[10] Click on the plus signs next to the individual items to expand the list.

Add the tables you need to your report by selecting them and clicking on the Add Table button. Double-clicking on the table will also add it to the list. When all have been added, go to the Links tab to establish the relationship between the tables.

[9] You should only use System DSN's. File DSN's and User DSN's are buggy.

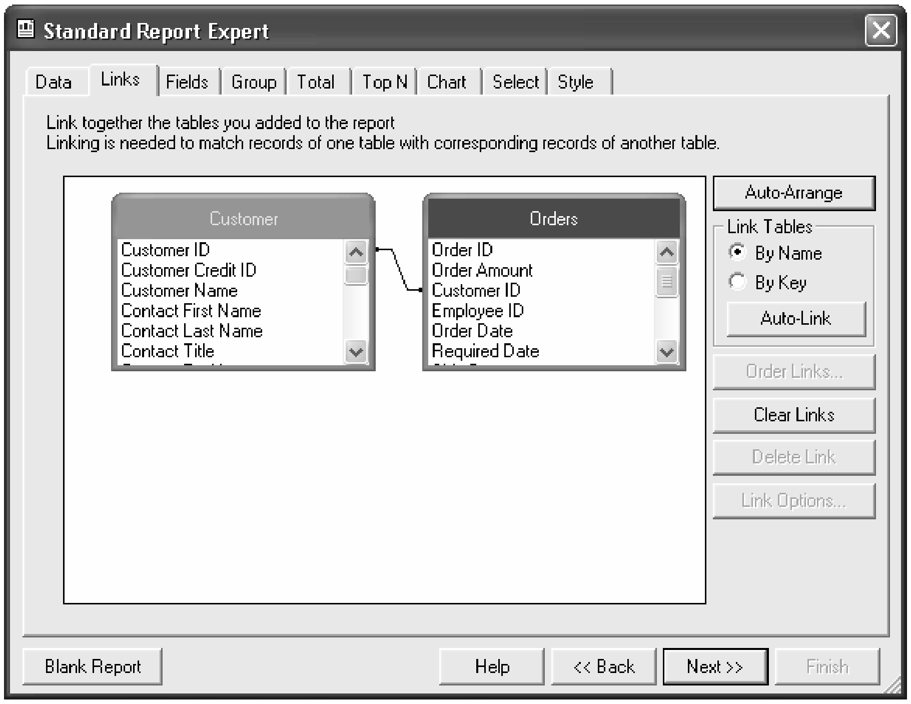
[10] If you don't see everything listed in the Database Expert (particularly stored procedures), right-click the report and select Designer | Default Settings. Go to the Database tab and click on the items you want to see in Database Expert.

Tip

There is one small quirk about the Available Data Sources window you should be aware of. It occurs when you click on the plus sign to expand a node and a data source dialog pops up. Once you close the dialog box there is no option to open the dialog box again. You have to click on the minus sign to close the node and then click on the plus sign to expand it again. This triggers the dialog box to open again. After adding a data source to that node, this isn't an issue because there will now be an item listed as the first item and clicking on it lets you add another data source.

Linking Tables

Whenever there are two or more tables, they need to be linked so that Crystal Reports knows how they are related. For example, a pet store that wants to print a list of its products by Animal Id needs to build a report using an animal table and a product table. To match the product to the appropriate type of animal, both tables will be linked by the Animal Id. It would be impossible to determine which products are associated with which animal without this link. The Links tab, shown in Figure 13-2, sets the linking fields.



Link together the tables you added to the report
Linking is needed to match records of one table with corresponding records of another table.

Customer	Orders
Customer ID	Order ID
Customer Credit ID	Order Amount
Customer Name	Customer ID
Contact First Name	Employee ID
Contact Last Name	Order Date
Contact Title	Required Date

Link Tables: [✓ By Name] By Key

Figure 13-2. The Links tab.

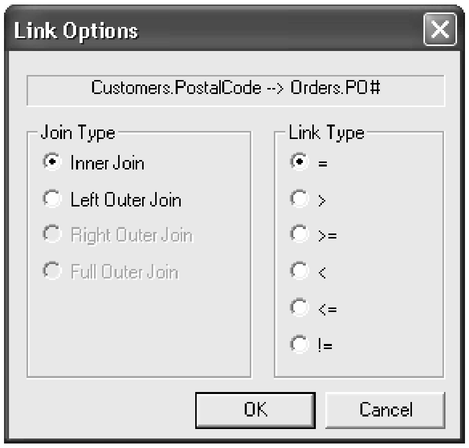
When the Links tab is first displayed, it creates default links. This is the equivalent of Crystal's "best guess" for the relationships between the tables. In an effort to make it easier for you, Crystal Reports tries to figure out which fields in each table should be linked to each other. It does this based upon indexes and fields that have the same name and data type.

Tip

To get the most benefit out of the auto-arranged links, design your tables with field names that use a consistent naming convention and have well thought out indexes. This will result in a higher probability that Crystal Reports will create the appropriate default links.

The default links are not set in stone. You are free do delete or add more, according to your needs. To delete a link, simply click on it (to select it); then click on the button labeled Delete Link. You can also just press the Delete key after selecting it. To add a new link, drag and drop the field from one table onto the matching field in the other table.

There are a couple of buttons that are helpful for managing links. The Auto-Arrange button rearranges the tables into an easier-to-read layout, which is useful when handling a report with many tables. A multitude of tables makes difficult the visualization of their relationship with one another and the overall structure. The Auto-Link button rebuilds the links based on whether you want to link by field name or by index. This comes in handy for undoing any new links you added, should you want to start from scratch. The Clear Links button removes all the links between the tables. The Link Options button opens the Link Options dialog box, shown in Figure 14 -3.



Join Type: [✓ Inner Join] Left Outer Join Right Outer Join Full Outer Join

Link Type: [✓ =] > >= < <= !=

Figure 13-3. The Link Options dialog box.

The Link Options dialog box establishes the type of relationship between the two fields. It sets the type of join (Inner, Left Outer, Right Outer, or Full Outer) as well as how the fields are compared (equal to each other, less than, etc.). Table 13-2 shows a list of the different linking options and how they affect the resulting data.

Table 13-2. Join Options

Join Type	Resultset Description
Inner Join	Records from the left table are matched with records from the right table. Only records with an exact match are included.
Left Outer Join	All records from the left table are included. Field values are included if there is a matching record in the right table . NULL values are stored in the corresponding fields if there is no matching record in the right table..
Right Outer Join	All records from the right table are included. Field values are included if there is a matching record in the left table. NULL values are stored in the corresponding fields if there is no matching record in the left table.
Full Outer Join	Every record from both tables is included. When records from both tables match, the fields in the new recordset are filled in as normal. The other fields are set to NULL if there is no matching record for one of the tables.

The Order Links button sets the order in which the links between the tables are created. This is used only when there is more than one link shown. The tables will be linked automatically, in the order that they are shown in the dialog box. The default linking order processes the links on the left before the links on the right. Changing the default linking order is useful when there is a hierarchy of tables joined together and the order to which they are joined is important. This can happen when you are using a link to return a subset of records from two tables, and these records must therefore be linked to another table. Changing the order of the links changes the resulting data.

Using Multiple Data Sources

Complex reports often require printing data of different origins. This can happen when you have tables in a SQL Server database and they need to be linked to tables in an Oracle database. There can also be SQL Server databases on different servers on the network. Although Crystal Reports is capable of printing such reports, it doesn't always run flawlessly. Using multiple data sources has the potential for a variety of problems. There is no easy fix.

Crystal Reports, by design, takes the data connectivity information that is saved with a report and generates SQL statements with it. The statements are then delivered to the database server. Although every major database vendor claims to be compliant with the SQL standard, they each have their own subtle differences that can cause them to reject the SQL language of another database. But Crystal Reports doesn't have the capability to, within a single report, generate SQL that is compatible with multiple variations of the SQL specification. It will inevitably create non-compatible SQL for one of the databases. You will have a compatibility issue.

That is not all. Different database servers manage their standard data types differently, chiefly in the way they store them internally. The way one database represents an Integer can be totally different from another database. When Crystal Reports tries to link these two fields together, it won't be able to because they look like different data types.

While it is true that Crystal Reports is designed to work with the majority of database servers, it doesn't imply that the database servers themselves will work with one another. Since every database tries to be better than the next database, companies are more concerned about performance than compatibility. There are limitations with reports using different databases.

Fortunately, there are some general rules you can follow to reduce the headaches caused by these problems. You should be able to generate the reports you need if you follow the following rules (with a little trial and error).

When linking tables together, only use fields that are of the String data type. String is the most consistent data type among different database servers and has the least likelihood of causing problems. This is no guarantee against all problems, however. For example, a string can be represented by variable and fixed length data type and this can create incompatibilities.

Rather than try to link tables from different data sources together, use subreports to print the same data. The benefit of using subreports is that they are treated individually. When Crystal Reports generates the SQL statements for the sub-report, it does so independently of the main report. Also, it has only to worry about working with a single database driver. This independence eliminates a lot of potential problems. Using subreports also allows for more flexible linking options between a main report and subreport. You aren't limited to linking with strings and you can also use formulas to perform data type conversions when necessary.

There are reports where using a subreport isn't an option. For example, if all the detail fields need to be printed side by side, you can't use subreports because the fields will have to be printed underneath the others. In this circumstance, find out if your database lets you create links to outside databases. Putting the links within the database server itself puts the responsibility of managing this data within the server. Crystal Reports benefits because now it only has to use one database driver to retrieve the data. You also get better performance because the database server maintains the additional connection and links.

Secured Databases

Most databases have security implemented in them. They require valid user credentials before accessing the data. This means that you have to pass a User Id and Password to the database prior to printing the report. Crystal Reports handles security differently, depending upon how the report is bound and the database used.

In most circumstances, a Windows application displays a login dialog box prompting the user to enter their user credentials. After entering the credentials, the user can login. However, this isn't always the case. If you are using an MS Access database with a database password or if you are printing from an Excel spreadsheet using OLE DB, the login dialog box isn't compatible. If you are writing an ASP.NET application, it doesn't have the capability to prompt the user for their credentials. In both of these examples the user won't be able to print the report. Rectifying this problem requires passing the user credentials during runtime. See Chapter 17 for a thorough discussion of connecting to every type of data source during runtime.

Caution

When using integrated security with ASP.NET applications, the SQL Server database should be on the same server as the web server. There are known issues when IIS tries to connect to SQL Server and then Crystal Reports tries to connect again. It can't perform a "double-hop" and the connection fails. See Microsoft Knowledge Base article 176377 for more information.

Connecting with Stored Procedures

As mentioned earlier in the chapter, stored procedures can be used as a data source just like a table. Crystal Reports can open a stored procedure, retrieve the data and print it. The one thing that can make a stored procedure unique is if it has input parameters.

For a stored procedure to execute, it must have a value for every input parameter. Crystal Reports automatically creates report parameters when it sees that the stored procedure has one or more input parameters. There is one report parameter for every stored procedure parameter and they will have identical names and data types. When the report runs, the user is prompted to enter a value for each parameter. Internally, Crystal Reports passes these report parameters to the stored procedure.

The following code is a sample stored procedure from the Northwind database.

```
CREATE PROCEDURE CustOrderHist
@CustomerID nchar(5)
AS
SELECT ProductName, Total=SUM(Quantity)
FROM Products P, [Order Details] OD, Orders O, Customers C
WHERE C.CustomerID = @CustomerID
AND C.CustomerID = O.CustomerID
```

In this stored procedure, there is one input parameter called @CustomerId that is a 5 character string. When this report is selected as the data source for a report, Crystal Reports automatically creates a parameter called @CustomerID as a String data type. The report prompts the user to enter a Customer Id when it is ran. The report passes this value to the stored procedure; the stored procedure only returns records with a matching Id.

Note

If you want your application to directly pass the parameters to the stored procedure, you have to set the parameter objects during runtime. The user won't be prompted with the parameter dialog boxes and the report will run seamlessly. See Chapter 17 for the steps to set report parameter values during runtime.

If you are using two stored procedures and linking them together, make sure that the parameters have different names. When Crystal Reports creates parameters for each stored procedure, it doesn't have the capability to create a new alias for the parameter names. Consequently, it will only create one report parameter with that name and it won't know which stored procedure to assign the parameter to. Make sure that parameters use unique names.

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).

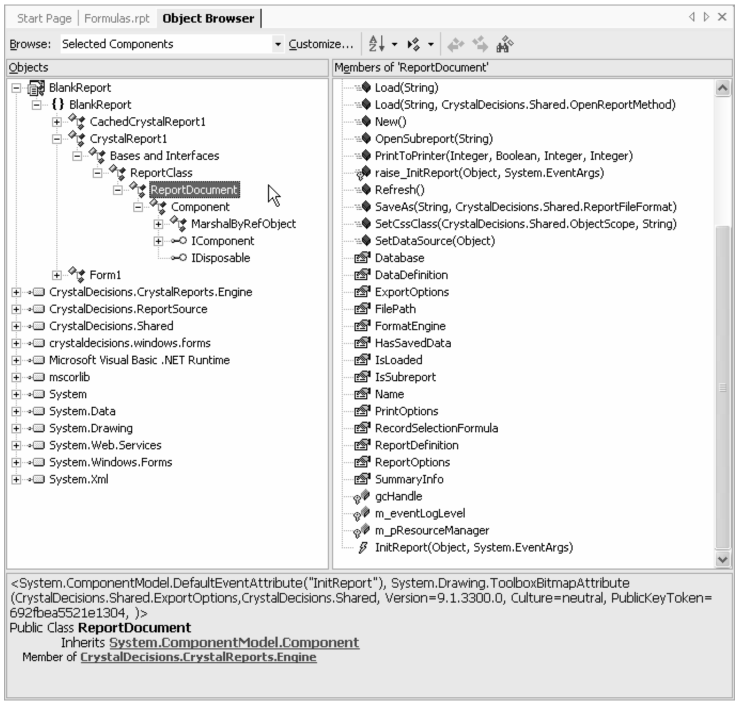
ASP.NET Template

Printing reports within an ASP.NET application requires a slightly 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.

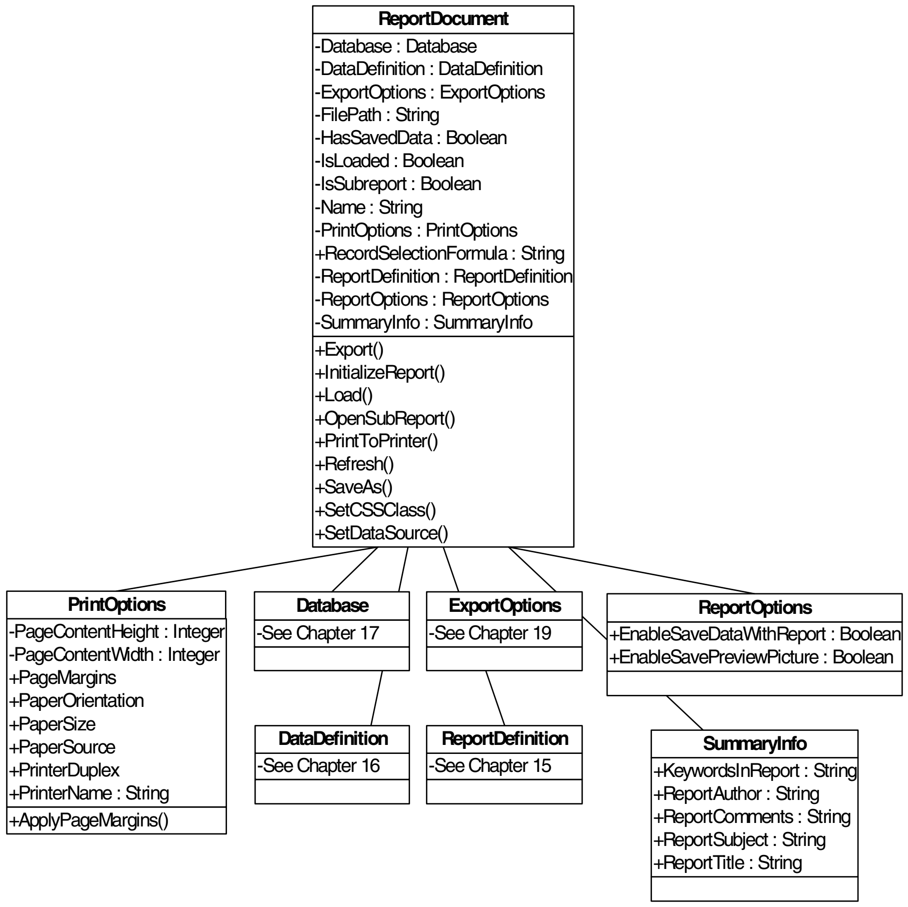


Figure 14-2. The ReportDocument object model.

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.

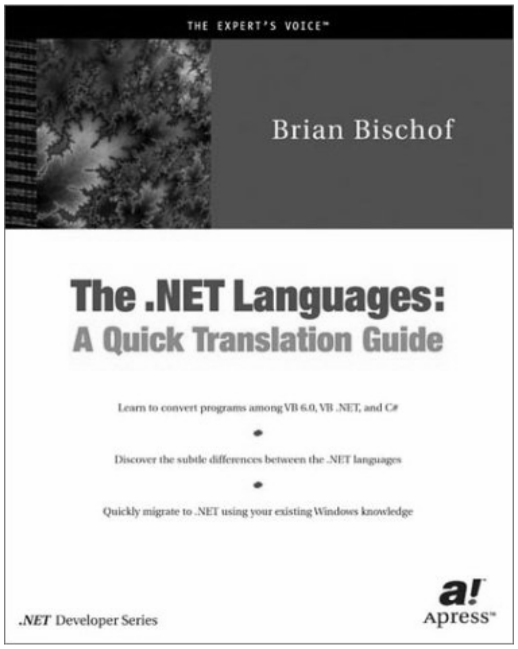
Property	Description
Exception	The class of exception that was raised.
Handled	Set to True if you do not want to the error triggered.
UserData	Overrides the error message.

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
```

Also by Brian Bischof

The .NET Languages: A Quick Translation Guide



THE EXPERT'S VOICE™

Brian Bischof

The .NET Languages: A Quick Translation Guide

Learn to convert programs among VB 6.0, VB .NET, and C#

•

Discover the subtle differences between the .NET languages

•

Quickly migrate to .NET using your existing Windows knowledge

.NET Developer Series

Apress™

If you are looking for a quick way to translate code between VB.NET and C#, this book is just what you need. Reviewers have praised "The .NET Languages" for its no-nonsense writing style that is so uncommon in today's computer books. Concise and to the point so you can take programming code and convert it to either .NET language without wasting time. The jewel of this book is the translation tables found at the beginning of each chapter. Every syntax keyword is clearly listed with a one-to-one mapping between languages. You'll keep this book close to your computer for these tables alone. The remainder of each chapter follows up with explanations for avoiding the programming traps you can fall into if you aren't careful. As a bonus, VB6 programmers will find that the entire VB6 syntax is included so that you can upgrade your skills to .NET in record time.

Buy "The .NET Languages" at bookstores worldwide and on Amazon.com (ISBN 1-893115-48-8). Also available in Italian (ISBN 88-8331-395-X) and French (ISBN 2-212-11107-X).