Programmer to Programmer ™

Crystal Reports is one of the world's leading software packages for creating feature-rich reports, and now it is fully integrated with Visual Studio .NET. With this book, you'll see how to produce elegant and effective reports for Windows and the Web.

Professional Crystal Reports for Visual Studio .NET

David McAmis

Wrox technical support at: support@wrox.com
Updates and source code at: www.wrox.com
Peer discussion at: p2p.wrox.com

What You Need to Use This Book

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

Software

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

Knowledge

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

Summary of Contents

Introduction		**1**
Chapter 1:	Crystal Reports.NET Overview	**9**
Chapter 2:	Getting Started with Crystal Reports.NET	**33**
Chapter 3:	Report Integration for Windows-Based Applications	**79**
Chapter 4:	Report Integration for Web-Based Applications	**113**
Chapter 5:	Creating XML Report Web Services	**151**
Chapter 6:	Working with .NET Data	**167**
Chapter 7:	Formulas and Logic	**201**
Chapter 8:	Working with the Crystal Reports Engine	**233**
Chapter 9:	Distributing Your Application	**267**
Appendix A:	Troubleshooting	**285**
Appendix B:	Migrating Applications to Crystal Reports.NET	**295**
Appendix C:	Crystal vs. Basic Syntax	**309**
Index		**315**

4

Report Integration for Web-Based Applications

While Windows applications will still continue to be popular for some time to come, the growing trend in application development is for web-based applications.

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

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

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

Obtaining the Sample Files

All the example reports and code used in this chapter are available for download. The download file can be obtained from www.wrox.com. Once you have downloaded the files, place them in a folder called CrystalReports\Chapter04 on your hard drive.

In Chapter 4, all of the completed projects are included in the downloadable code as well as the reports used throughout the chapter, so you can either browse through the finished projects or create your own projects from scratch using the components provided.

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

Planning Your Application

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

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

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

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

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

The options for working with reports are endless – based on a user's access rights in your application, you could set a specific record selection formula or allow the user to set and retain parameters they use frequently, or even establish profiles of their favorite reports, so they can run it with all of their settings in place with one click.

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

A Brief History of Crystal Web Development

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

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

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

With the release of version 8.0, the reporting technology took another massive leap forward, but some of the same limitations persisted (such as scalability and security) until the advent of Crystal Reports 8.5 and the introduction of Crystal Enterprise 8.5. Leveraging the architecture and code base from Seagate Info, Crystal Enterprise provides a robust application framework that developers can use to create applications that can be scaled from one to ten to ten thousand users and beyond.

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

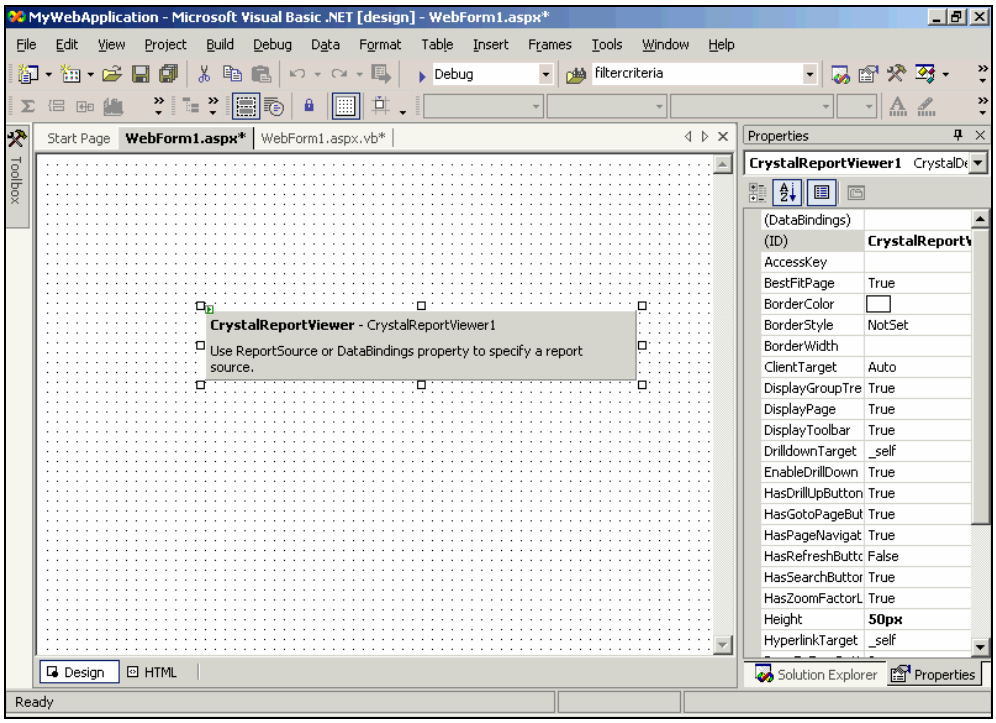
> *Note: To deploy applications beyond a workgroup implementation of 5-10 users to a large number of users you will need to purchase an additional license from Crystal Decisions. Also, if you need to offload processing in a true n-tier application or want to schedule or redistribute reports, you may want to consider moving your application to Crystal Enterprise (covered in Chapter 9).*

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

Exploring the Development Environment

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

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



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

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

Using the object models provided by Crystal Reports.NET, you have almost complete control over the report's appearance and behavior.

Before You Get Started

Before we can actually get into creating web-based applications, you will need to check and see if you have all of the required components installed to run these applications. ASP.NET web applications run on a web server that can either be located on your local machine or on another server that you have access to that has IIS installed.

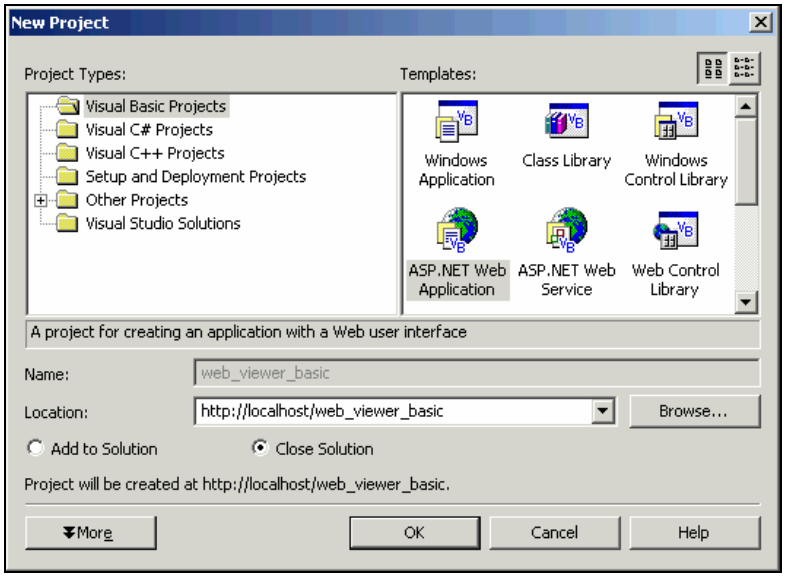
When you installed Visual Studio .NET, you may have received an error message if you did not have a web server installed on your machine at that time. If you are working on a computer that does not have IIS installed and the required .NET components loaded, you will need to have access to a server that does in order to create the forms and applications demonstrated in this chapter.

For more information on installing the .NET Framework and preparing a web server for application development, check out the Visual Studio .NET Combined Help Collection and search for "Configuring Applications".

Starting a New Web Application with VB.NET

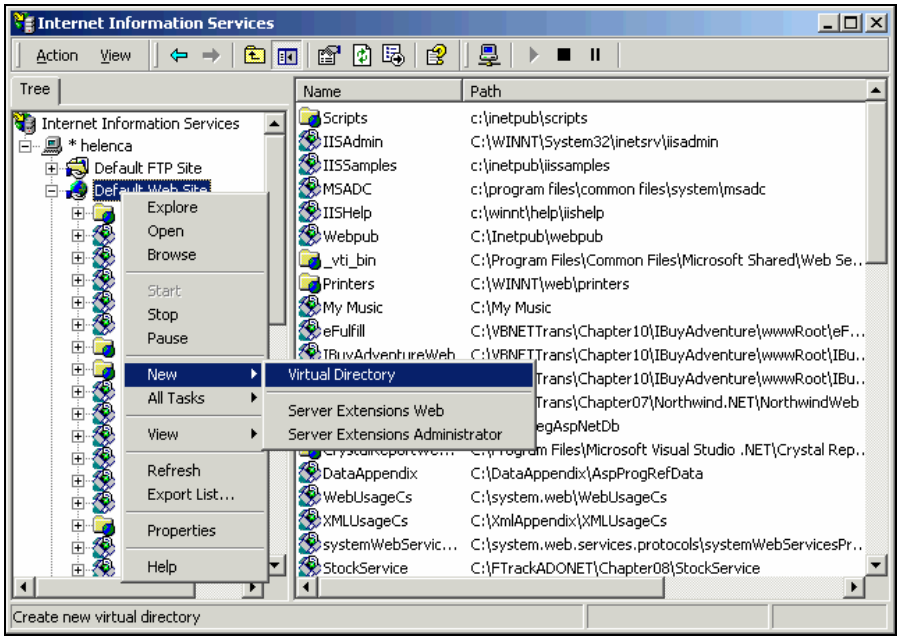
The first thing we need to do to get started is to create a new web application using Visual Basic .NET. Included with the download files for this chapter are a number of projects that are related to the different sections in this chapter. To walk through the examples that follow, you can either create a new solution or open the one that is provided (the same applies to the other projects – you can either follow along or create your own).

To create a new web application, from within Visual Studio, select File | New | Project and from Visual Basic Applications, select ASP.NET Web Application and specify a name (`web_viewer_basic`) and location for your project files.



Since you are creating a web application, the location will be a web server that you have access to and the name of your project will actually be used to create a virtual directory on this server. (The good news is that Visual Studio .NET will automatically do this for you if you are building the application from scratch – there is no need to create the folder and virtual directory prior to creating a new project.)

If, however, you choose to use the supplied download code, then you should create a virtual directory (in our case, this is `C:\CrystalReports\Chapter04\web_viewer_basic`) by selecting Control Panel | Administrative Tools | Internet Information Services, and then right-clicking on Default Web Site. This will open another menu:



Select New | Virtual Directory and the virtual directory wizard will commence. Assign the new directory the alias `web_viewer_basic` and set the path to `C:\CrystalReports\Chapter04\web_viewer_basic`. Make sure both read and write are enabled and finish the wizard.

Either way you choose to do it, the development environment will open with a default form that we will be using in the section. Throughout the chapter, we will be using only one or two Web Forms to demonstrate different integration features, but the same concepts can be applied to your own web applications.

Before you go any further, we need to get some basic architecture decisions for your web application out of the way, starting with a brief discussion of the object models available within Crystal Reports.NET.

Determining the Correct Object Model

When working with web applications, there are two different object models to choose from, each with its own capabilities and strengths. The first, contained within the **Crystal Reports Web Forms Viewer** object model (`CrystalDecisions.web`), contains all of the functionality required to view a report in the Crystal Reports Web Forms Viewer, including the ability to set database logon information, pass parameters and record selection, control the viewer's appearance, and view reports, including reports consumed from an XML Report Web Service.

Using the `CrystalDecisions.Web` object model, you are covered for most basic report integration requirements, but you have no control over the report itself at run time – you won't be able to change the record selection for any subreports that appear in your report and you won't have access to modify report elements, like groups and sorting, or formula fields.

For complete control over the report and its content, you need to use the **Crystal Reports Engine** object model (`CrystalDecisions.CrystalReports.Engine`) in conjunction with the viewer object model. This will allow you complete control over your report and the objects and features contained within. Using the Crystal Reports Engine means that you have a rich object model that can be used to modify even the tiniest elements of your report.

> *Note: You will also need to use the Report Engine object model if you are using ADO (.NET or "Classic" ADO) as the data source for your report (which is covered in Chapter 6:* Working with .NET Data*).*

It is important to note that the Crystal Reports Engine object model cannot stand alone – it provides no way to view a report and relies on the Crystal Reports Web (or Windows) Forms Viewer to actually view the report.

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

Understanding the CrystalDecisions.Web Namespace

The `CrystalDecisions.Web` namespace contains all of the classes that relate to functions available within the Web Forms Viewer itself. The following table illustrates the different classes that are available, as well as their use in web applications.

Class	Description
`CrystalReportViewer`	Contains the properties, methods, and events relating to the `CrystalReportViewer` and viewing reports. Note: some properties of this class are inherited from `CrystalReportViewerBase`.

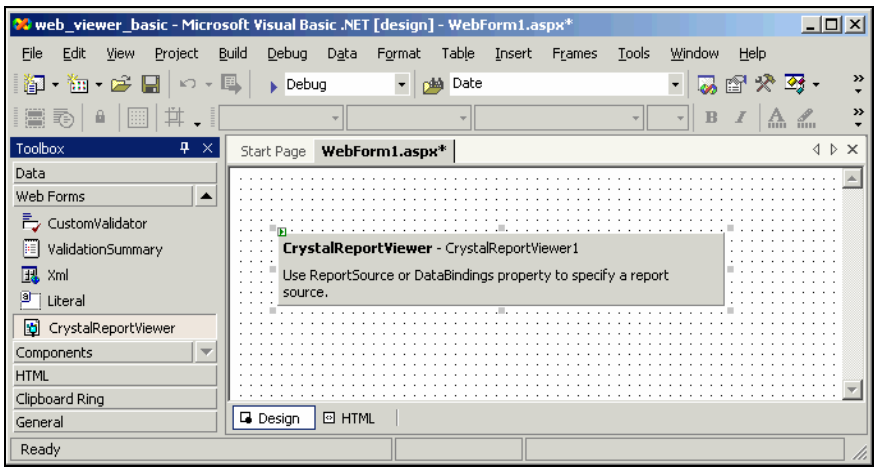
Table continued on following page

Class	Description
`CrystalReportViewerBase`	Contains properties for setting the target browser edition, database logon information, etc.
`DrillEventArgs`	Provides data for the `Drill` event on main reports and subreports. `Drill` events fire when a user drills down into a group or summary on a particular main report or subreport.
`DrillSubreportEventArgs`	Provides data for the `DrillDownSubreport` event on main reports and subreports. `Drill` events fire when a user drills down into a group or summary on a particular main report or subreport.
`ExceptionEventArgs`	Provides data for the `HandleException` event. `HandleException` events occur when there is an error or exception when setting report properties or viewing the report. Primarily used for troubleshooting and error messages.
`NavigateEventArgs`	Provides data for the `Navigate` event. When a user navigates through the pages of a report, the `Navigate` event fires each time. This can be used to notify the users when they have reached the last page, to call custom actions, etc.
`SearchEventArgs`	Provides data for the `Search` event. The `CrystalReportViewer` includes an integrated search function to search for values within a report. The `Search` event fires when a user searches for a value and could be used to trigger a search of another report or other report descriptions, etc.
`ViewerEventArgs`	Provides data for the `Viewer` event. The `Viewer` event fires when some action has occurred within the viewer itself and can be used to launch other actions when the viewer loads, etc.
`ZoomEventArgs`	Provides data for the `ViewZoom` event. The `ViewZoom` event fires when the zoom is changed on the viewer and can be used to suggest the best resolution for a particular report (100%, 200%, etc.), or if you are showing two reports in viewers side by side, to synchronize the zoom factor between the two (so the magnification on both reports stays the same – if you change one to 200%, the other view changes as well).

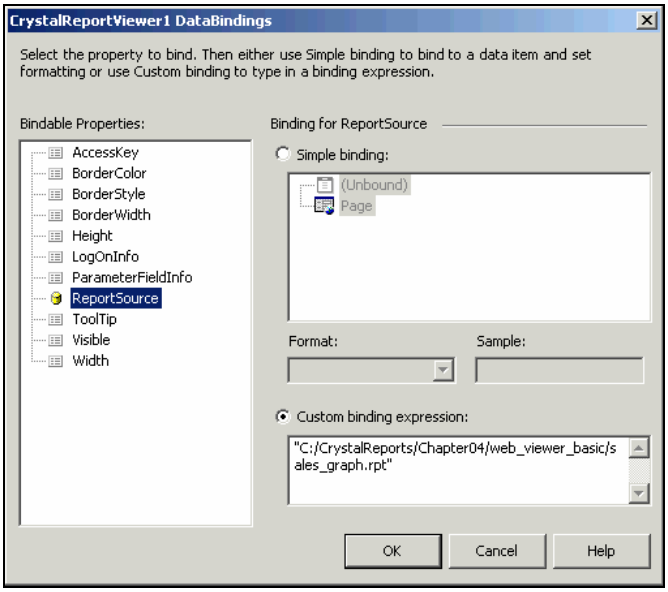
Using the Crystal Report Viewer for Web Forms

For report files that live externally to your application (for instance, as a standalone report file, created with either this or a previous version of Crystal Reports) there is not much to creating a simple preview form for your report. We are going to walk through that process in the following section.

Earlier we created a new project called `web_viewer_basic` and within that project there should be a default Web Form (`WebForm1.aspx`) that was created when you created the project. To start, we need to drag or draw the Crystal Report Viewer onto our Web Form:

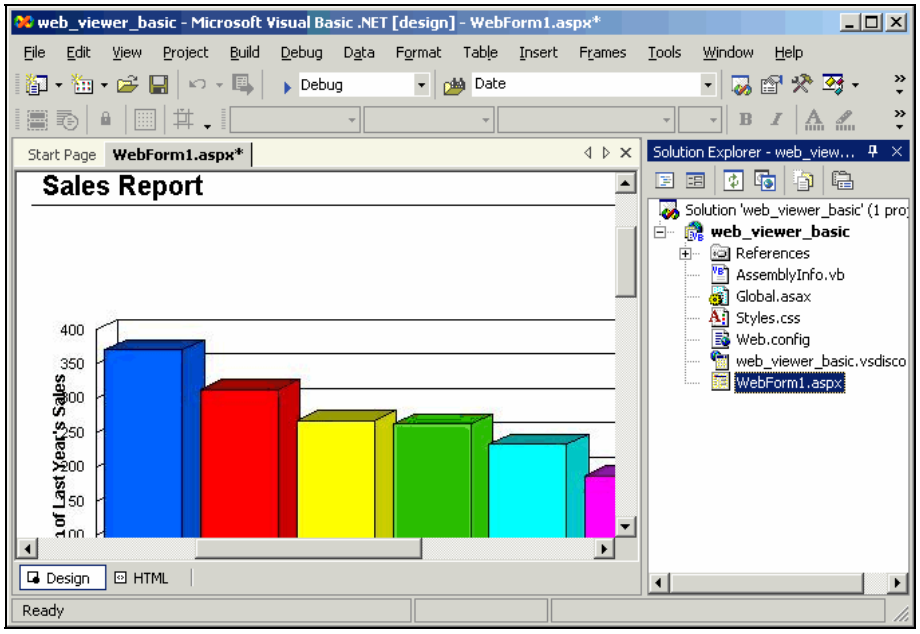


From that point, we need to set the ReportSource property to let the viewer know where to get the report from. To access this property, locate the Properties window for the Crystal Report Viewer and open the (DataBindings) property, by clicking on the ellipse at the side, to show the dialog that appears below:



Click on the ReportSource property and click on the radio button to select Custom Binding Expression. In this example, we are going to assume that you have unzipped the download files for this chapter to your hard drive in a folder called `CrystalReports\Chapter04` – included in these files is a Sales Graph report (`sales_graph.rpt`) that we will be using through this walkthrough.

Once you have entered the name and path for your report, click OK to accept your changes and return to the form we were working with. The report will now be displayed in a "Preview" mode in the form designer, as shown here:

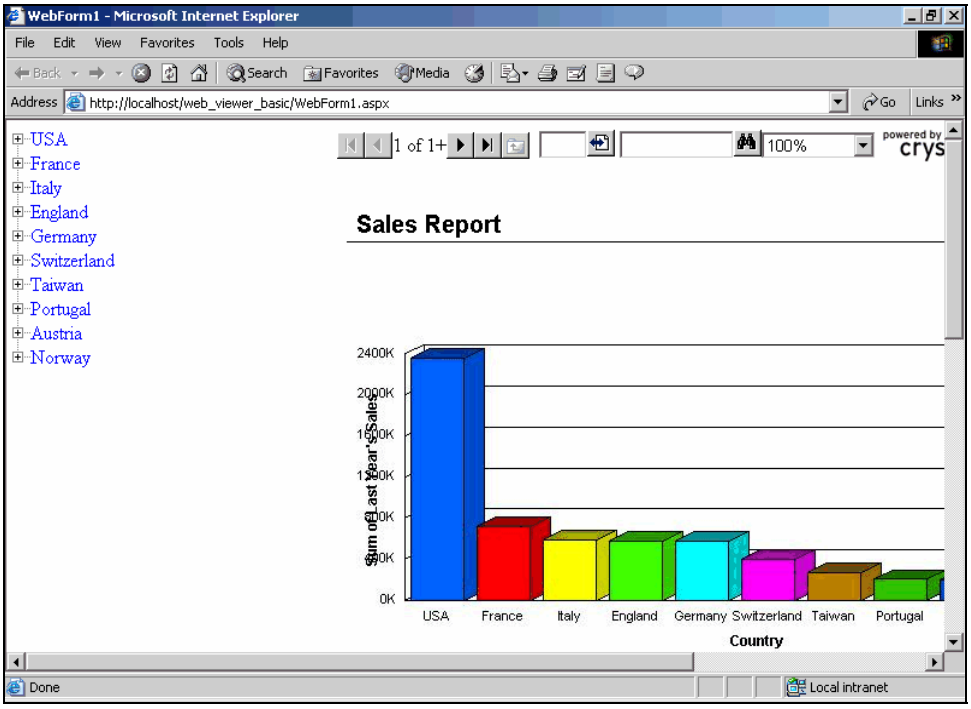


If we were working with a Windows form, this would be all that is required to actually preview a report. Since Web Forms work a little differently, there is an extra step involved before we can run our application and preview our report.

Double-click anywhere on your form to open the code view for the form and locate the section of code marked Web Form Designer generated code. Expand this section to find the `Page_Init` function and add a line of code immediately after the `InitializeComponent()` call to bind your report to the viewer when the page is initialized:

```
Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Init
  'CODEGEN: This method call is required by the Web Form Designer
  'Do not modify it using the code editor.
  InitializeComponent()
  CrystalReportViewer1.DataBind()
End Sub
```

Whenever you run your application and preview the form, your report will be displayed in the Crystal Report Viewer, as shown next:



The viewer interacts with the Crystal Reports print engine, runs the report, and displays the results. From your report preview, you can drill-down into the details or search for a value, without having to do any additional coding.

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

In the following sections, we are going to walk through adding a report to your application, binding the report to the Crystal Report Viewer, and customizing the viewer.

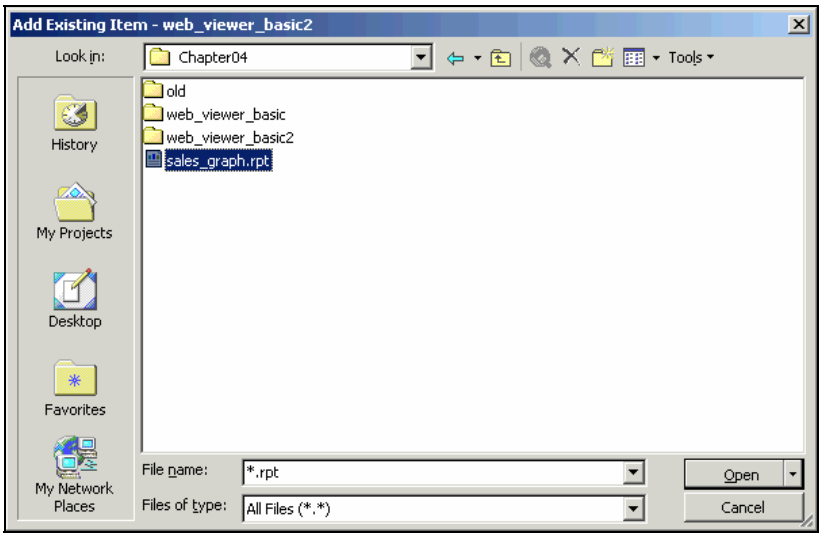
Adding a Report to Your Application

To add a new report to your application, you have two choices – you can either use an existing report that you have created (using this or a previous version of Crystal Reports), or you can use the Report Designer integrated within Visual Studio .NET to create a report from scratch. For our purposes, we are going to add an existing report to our next sample application (for more information on creating reports from scratch, check out Chapter 2: *Getting Started with Crystal Reports*.)

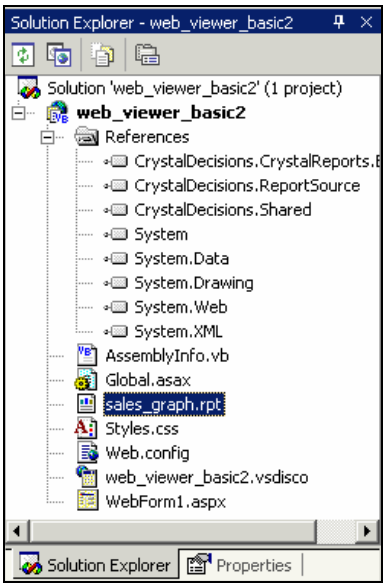
In this example, we will add the Sales Graph report to `web_viewer_basic2`. Whereas the approach in our first example favors publication of a report that will always be found by the same path, but is subject to regular updates, this option favors a report that is not likely to change and so can be incorporated into the application, allowing us to alter its features, or even build it from the ground up. This is also relevant to whether we are using strongly typed or untyped reports, as we discussed in the last chapter.

Once again, you have the choice of building the project or using the code provided – but remember that if you use the code provided, you must create a virtual directory for it in IIS on your machine.

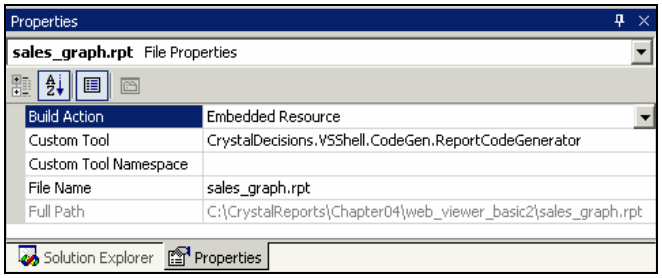
To add our Sales Graph report to this new project, select Project | Add Existing Item, which will open the dialog shown below. Change the drop-down list to show All files and specify *.rpt for the file name to filter the list to show only the available reports.



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



Once you have added your report to your project, it will appear in the Solutions Explorer and you can view the design of the report using the Report Designer and access its properties through its Properties page.



Adding the Report Viewer to a Web Form

You can add the Crystal Report Viewer from the Web Forms Toolbox and drag or draw the viewer onto your form – unlike the Crystal Report Viewer for Windows forms, the Web Forms Viewer does not provide a view of how the viewer will appear on your page until you actually bind a report to it, at design or run time.

In fact, we'll do this now for `web_viewer_basic2`. Just drag a CrystalReportViewer over from the Toolbox and place it on the Web Form. Don't bother setting a report source for it yet, as we are about to look at different methods for binding the report to the viewer.

The Crystal Report Viewer can be used on existing forms to display a report side by side with other controls, or you could add the report to a new form to have a separate window for previewing reports.

Binding a Report to the Report Viewer

With the Report Viewer added to your form, we now need to bind a report to the viewer itself. As we saw in Chapter 3 for Windows-based applications, there are five different ways to bind a report to the Crystal Report Viewer:

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

Binding by Report Name

To bind a report using the report name, as we did in our first example, all you need to do is set the `ReportSource` property, either through the Data Bindings properties for the report or through the form's code, as shown here:

```
CrystalReportViewer1.ReportSource =
"C:\CrystalReports\Chapter04\web_viewer_basic\sales_graph.rpt"
```

Then in the `Page_Init` event of your form, you would need to add a single line of code to call the `DataBind` method immediately after the `InitializeComponent()` call, as shown here:

```
CrystalReportViewer1.DataBind()
```

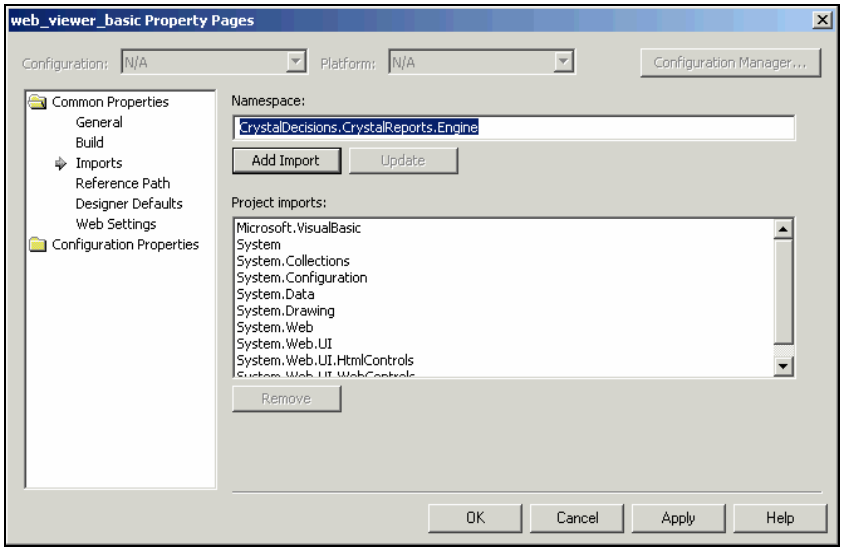
If you prefer to set the initial report source using the property pages, you will need to open the Properties page for the report viewer, and then select the (DataBindings) property as we did in the first example.

But since we have added the report to our application, we could also bind the report by creating a report object, loading the report into the object, and binding the object to the Crystal Report Viewer.

Binding by Report Object

Binding by a report object works slightly differently depending on if you have added the report to your project, or if you are referencing it externally. For a report that resides external to your application, you need to first specify an import of the `CrystalDecisions.CrystalReports.Engine`, which will allow you to create the object.

If this is not already present, we shall add it. In the Solution Explorer, right-click on your project title and select Properties from the right-click menu to open the dialog shown below.



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

With this namespace imported, we now need to create the report object as a `ReportDocument` in the form's `Declarations` section, as shown:

```
Dim myReport as New ReportDocument()
```

With our object now ready to be used, we can now load the external report file by placing the following code under the Page_Init event:

```
...
InitializeComponent()
myReport.Load("C:\CrystalReports\Chapter04\sales_graph.rpt")
CrystalReportViewer1.ReportSource = myReport
```

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

```
...
InitializeComponent()
myReport = New sales_graph()
CrystalReportViewer1.ReportSource = myReport
```

or, if you wanted to eliminate the `myReport` variable:

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

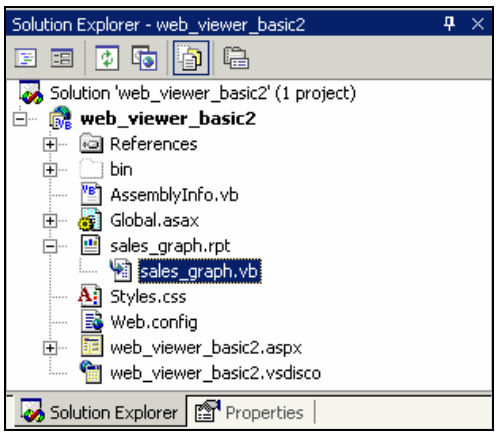
Which method you choose will depend on where the report is physically located and how you wish to access it.

Binding to an Untyped Report

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

When integrating reports into an application, we can either use strongly-typed reports or untyped reports. If you have been working with Visual Studio .NET for any length of time, you have probably heard of strongly-typed objects. A strongly-typed object is predefined with a number of attributes that are specific to that object, giving programmers more structure and a rigorous set of rules to follow, whereas an untyped report does not have this structure or rules applied to it, making it more difficult to work with.

Within the frame of reference of Crystal Reports.NET, a strongly-typed report can be any report that has been added to your project. When you add a report to a project, you will notice that in addition to the report, another file (with a `.vb` extension) is also added, as shown overleaf:



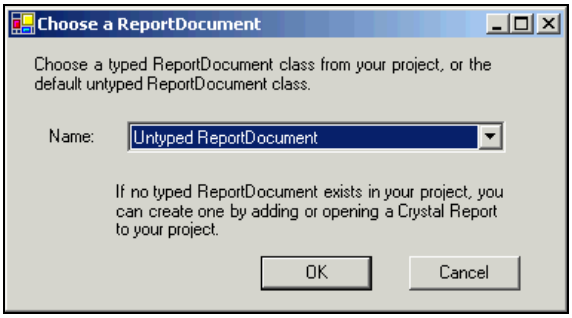
Note: This file is hidden until you select Show All Files *from the Solution Explorer.*

This additional file is the report source file and contains a report class specific to each report, called `ReportDocument`, which is derived from the `ReportClass` class and is created automatically for you.

An example of an untyped report would be a report that is stored externally to your project. For instance, you could view a report by setting an external reference (as in our first example, where we set the ReportSource property in the (DataBindings) to `"C:\CrystalReports\Chapter04\web_viewer_basic\sales_graph.rpt"`.

We'll just have a brief look at how we do this. Create a virtual directory called `web_viewer_untyped` pointing at `C:\CrystalReports\Chapter04\web_viewer_untyped` (if this is where you have downloaded the source code to) or alternatively build it from scratch.

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

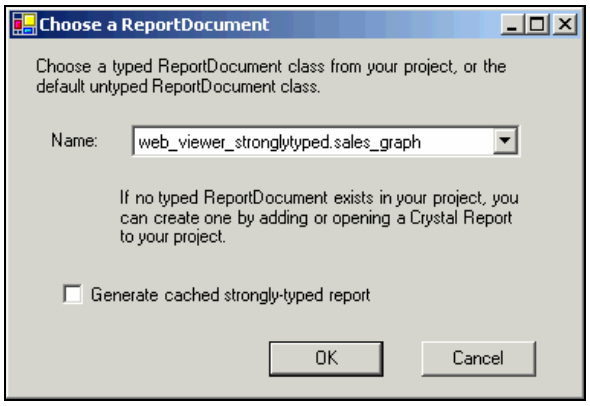


It is using this dialog that we set whether our ReportDocument component is typed or untyped. If you select Untyped ReportDocument, then we are not really accomplishing much new here. Drag a new `CrystalReportViewer` onto the Web Form, and then load a report into ReportDocument1 and bind the component to the viewer.

```
...
InitializeComponent()
Dim ReportDocument1 As New ReportDocument()
ReportDocument1.Load("C:\CrystalReports\Chapter04\sales_graph.rpt")
CrystalReportViewer1.ReportSource = ReportDocument1
```

Binding to a Strongly-Typed Report

Finally, you can choose to add a strongly-typed report component, which probably has the simplest binding method of all. First of all, add the report that you wish to bind to your project. In our case, this will be `sales_graph.rpt`. To create a strongly-typed ReportDocument component, drag the ReportDocument component onto your Web Form. (The code for this Web Form is available in the code download for the chapter as `web_viewer_stronglytyped`.)



You will then see the same dialog before, with a drop-down list of all of the available reports that are in your project. Select an existing report to create a strongly-typed ReportDocument. Now, insert the `CrystalDecisions.CrystalReports.Engine` namespace into the project using the Properties page as we did previously, drag on a `CrystalReportViewer`, and we're set. From that point, we just need to set the ReportSource property in the `Page_Init` event once more:

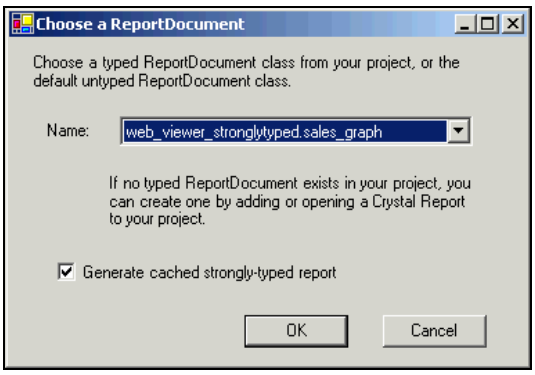
```
CrystalReportViewer1.ReportSource = sales_graph1
```

(Where `sales_graph1` is the name automatically assigned to the ReportDocument component when you added it to your form.)

Binding to a Strongly-Typed Cached Report

Another option for strongly-typed reports is the ability to use ASP.NET caching with your report. When you added your report to your application, you may have noticed that there was a checkbox for Generate cached strongly-typed report. Report caching is based on the underlying ASP.NET caching model and provides an easy way to improve your application's performance.

When a report is cached, the subsequent Web Form that is used to view the report will load faster when accessed by different users. To add a report to your Web Form as a cached report, select this option as you add a strongly-typed report to your application from the dialog shown overleaf:



You will see that your report file will be inserted as `cached_sales_graph1` and an additional object will be inserted into the Web Form's source file.

You could then bind to this particular cached report just as you would to any other strongly-typed report, but with a different name:

```
CrystalReportViewer1.ReportSource = cached_sales_graph1
```

When multiple users visit the same report, they will actually be looking at a cached copy and not hitting the database in real time. (To ensure this is true, the viewer by default does not have a refresh button showing.)

So regardless of which method you choose to bind your report to the viewer, the result is the same. For ease of use and functionality provided, the easiest method is going to be to stick with strongly-typed reports, because in the long run the structure and coding standards will mean you can create reporting applications quickly, in a consistent manner.

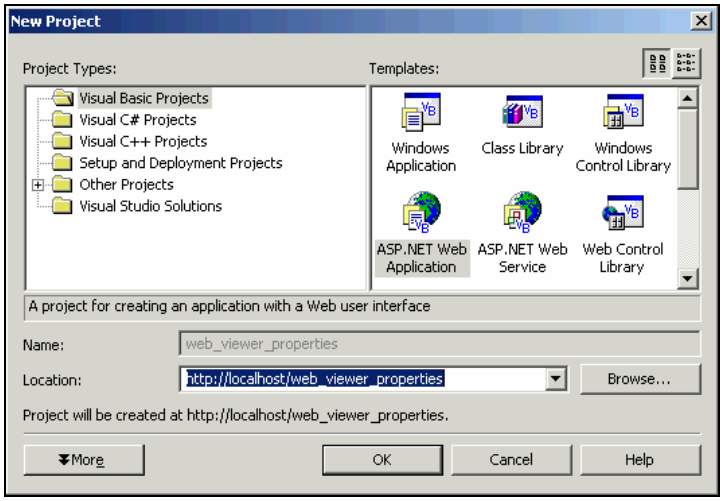
After binding, you can run your application and when the form is loaded the Crystal Report Viewer will run the report you have set in the ReportSource property and display a preview of it. But before we can move on to customizing the viewer, we need to look at working with secured databases.

Note: Before we finish up with viewer basics and binding, keep in mind that reports that were created from a secure data source may require a user name and password. Turn back to Chapter 3 to the section titled "*Passing Database Logon Info*" to review the use of the `LogonInfo` collection – it behaves in the same way for either the web or Windows form viewers and the same goes for the record selection formula – it can be returned or set using the `SelectionFormula` property, and its use is also described in Chapter 3.

Customizing the Appearance and Layout of the Report Viewer

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

To get started, we need to create a new project to work in – from within Visual Studio, select File | New | Project and from Visual Basic Applications, select ASP.NET Web Application and specify a name and location for your project files.



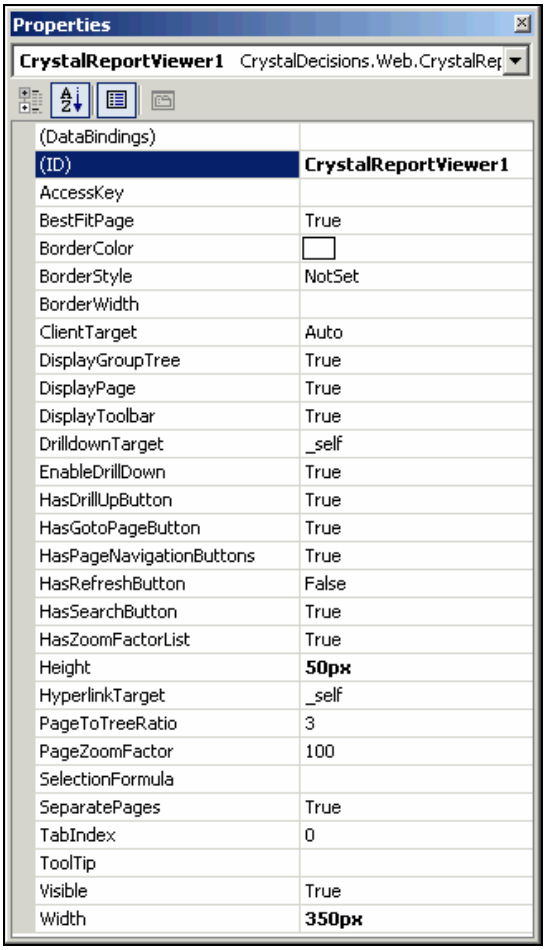
In the sample files, we have called this project (`web_viewer_properties`). Once you have selected a name for your project and clicked OK, the development environment will open with a default form that we will be using in the section. Alternatively, you can create a virtual directory for this project using the sample code provided.

We also need to add a report to work with in this section, so select Project | Add Existing Item and select `product_listing_bytype.rpt`. Add this to your project, insert the `CrystalDecisions.CrystalReports.Engine` namespace, drag across a `ReportDocument` and place this on the form (selecting `product_listing_bytype.rpt` out of the options on the dialog box), and then insert the `CrystalReportViewer` and set the binding to this report in the `Page_Init` event:

```
CrystalReportViewer1.ReportSource = product_listing_bytype1
```

You are now ready to get started!

When you were working through the earlier example, binding to a viewer and previewing your report, you may have noticed that there is a standard set of icons and layout that appears by default on the `CrystalReportViewer`. You can control most of the aspects of the viewer and toolbar by setting a few simple properties, as shown overleaf.



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

Property	Description
`BestFitPage`	For showing the report as-is or with scroll-bars
`DisplayGroupTree`	For showing the group tree on the left-hand side of the viewer
`DisplayPage`	For showing the page view
`DisplayToolbar`	For showing the entire toolbar at the top of the viewer
`SeperatePages`	For displaying a report in separate pages or one long page

All of these properties default to `True` and you cannot change the position of any of these elements – they are fixed in place on the viewer. You can, however, hide all of these icons and create your own buttons for printing, page navigation, and so on.

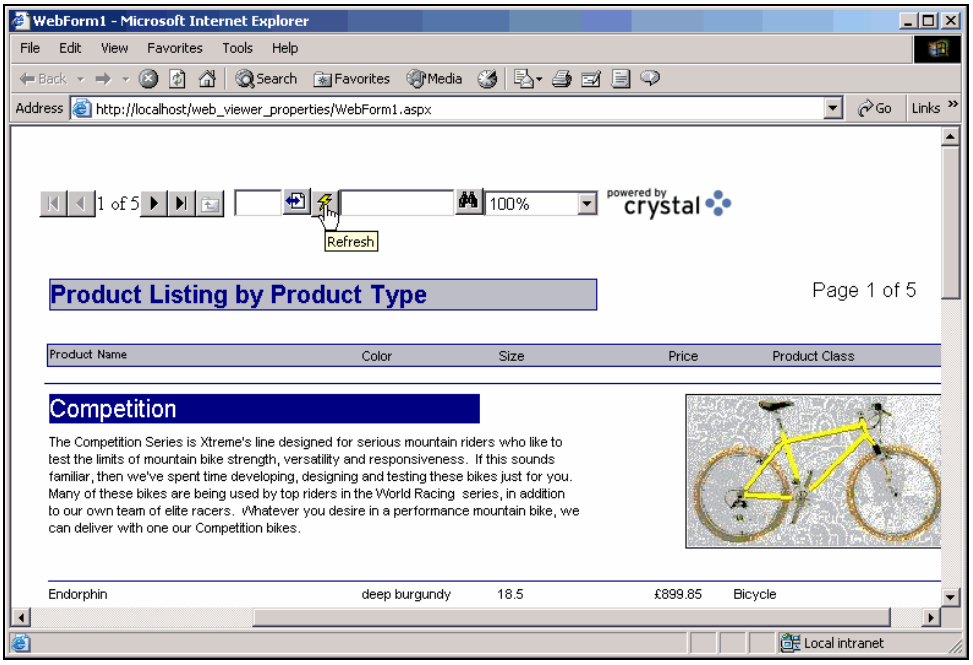
For the icons within the toolbar, you can also set simple Boolean properties to show or hide a particular icon, as shown below:

- `HasDrillUpButton`
- `HasGotoPageButton`
- `HasLevelUpButton`
- `HasPageNavigationButtons`
- `HasRefreshButton`
- `HasSearchButton`
- `HasZoomFactorList`

So a typical use of these properties is where you want to give users a preview of the report with the ability to refresh the data shown. You could easily set a few properties before you set your `ReportSource` property to make this happen:

```
CrystalReportViewer1.HasRefreshButton = true
```

When the report is previewed, it will appear as shown:



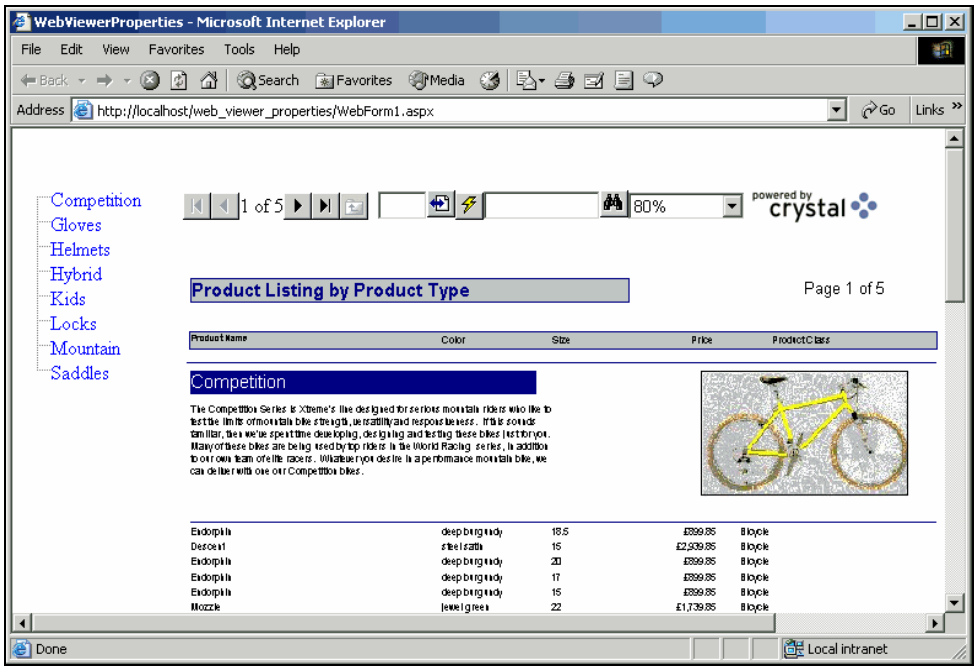
In addition to simple Boolean properties, there are also a couple of other properties that can be set to control the appearance and behavior of the viewer, including:

Property	Description
`PageToTreeRatio`	For setting the ratio between the group tree and the rest of the page – larger numbers mean a larger report page, with a smaller group tree
`PageZoomFactor`	The initial zoom factor for the report when viewed

So if we wanted to change the `PageToTreeRatio` and zoom factor so that the report was presented a little bit better on the page, we could add the following code to be evaluated when the page was loaded:

```
CrystalReportViewer1.PageToTreeRatio = 7
CrystalReportViewer1.PageZoomFactor = 80
```

Our previewed Web Form would look like this:



Viewer Methods

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

Create a new Web Form, which we'll call `web_viewer_methods`. Again, the code for this application is included with the download code. Drag a CrystalReportViewer onto this form. Include the report `product_listing_bytype.rpt` in your project (in the download code, the path is `CrystalReports/Chapter04/product_listing_bytype.rpt`). Drag a ReportDocument component from the Toolbox onto your form, and when the dialog box opens up, select web_viewer_methods.product_listing_bytype from the drop-down box. Click OK.

Now we add some code to tie our report to the application. In the `Page_Init` event in the designer generated code, once again add:

```
CrystalReportViewer1.DataBind()
```

Now all that remains is to set the `ReportSource` property in the `Page_Load` sub:

```
CrystalReportViewer1.ReportSource = product_listing_bytype1
```

Compile and run this. Now, we're all set to customize our viewer.

In this example, we are actually going to walk through building a custom viewer. The first thing we need to do is set the DisplayToolbar property and the DisplayGroupTree property to `False` in the Properties pane for the viewer, and add some additional buttons and textboxes to our Web Form using the screen shot earlier as a guide, which we will walk through below.

As we walk through this example, we are going to add the code behind these buttons and this form using the methods described below and learn how to match the viewer user interface to your own application.

Setting Browser Rendering

The `CrystalReportViewerBase` class provides a number of key properties, one of which is the `ClientTarget`. The `ClientTarget` property is a string and determines how the Crystal Report Viewer will render the report.

These strings are:

- `ie4` – for Internet Explorer 4.0
- `ie5` – for Internet Explorer 5.0
- `uplevel` – for most other web browsers
- `downlevel` – for very basic web browsers

A web browser is considered `uplevel` if it can support the following minimum requirements:

- ECMAScript (JScript, JavaScript) version 1.2.
- HTML version 4.0
- The Microsoft Document Object Model (MSDOM)
- Cascading style sheets (CSS)

Browsers that fall into the `downlevel` category include those browsers that only provide support for HTML version 3.2.

So, to set the browser version you are targeting, you could set the `ClientTarget` property for your form like this, under the `Page_Load` subroutine:

```
CrystalReportViewer1.ClientTarget = "ie4"
```

There is also an `Auto` value, which is the default setting and automatically selects the best rendering option based on the browser type. Unless you are writing an application for a specific browser or compatibility level, leaving this property set to `Auto` will provide the best viewing experience for the browser you are using.

> **For more information on detecting the browser type your web application is using, see the topic "*Detecting Browser Types in Web Forms*" in the Visual Studio .NET combined help collection.**

Refreshing the Data in a Report

When a report is refreshed, it goes back to the database for the most current set of data available and runs the report again. On our custom web viewer, you should have a Refresh button, so pull a Button control onto the Web Form and rename it `Refresh_Button` in the ID property in the Properties pane. Change the text property to `Refresh`.

Now, click on the `Refresh_Button` on your form to open the code for it. We can add some code behind this button to refresh the report using the `RefreshReport` method as shown below:

```
Private Sub Refresh_Button_Click(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles Refresh_Button.Click
  CrystalReportViewer1.RefreshReport()
End Sub
```

Compile and run the application. The button should now be present on your form. Click on it. This will cause the report to return to the database and read the records again. Use this functionality with caution – if a report has a large SQL query to perform before it can return the first page, you may experience performance problems.

Page Navigation and Zoom

Now we are going to insert some buttons across the top of our Web Form in the same way we did with the Refresh button, with the following names and text values:

Button Name (ID Property Value)	Text Property Value
`FirstPage_Button`	`First`
`Back_Button`	`Back`
`Forward_Button`	`Forward`
`LastPage_Button`	`Last`

We access these properties, once again, through the Properties pane in Visual Studio .NET.

For page navigation using the buttons we have drawn on our custom form, we have a number of methods that can be called without any additional parameters, as shown below:

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

These methods do not return a result, and unlike the Windows Forms Viewer, the Web Form Viewer does not have a `GetCurrentPageNumber` method, which would have returned an integer representing the page you are currently viewing.

To add these methods to the page navigation buttons, double-click the appropriate buttons on your Web Form and enter the code behind, as shown:

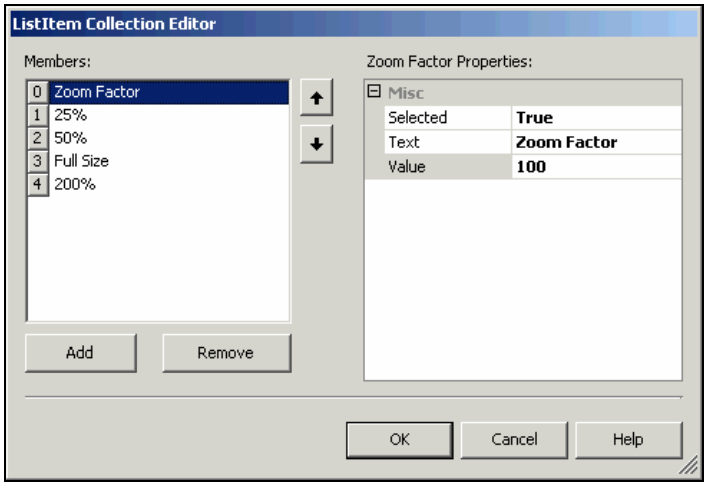
```
CrystalReportViewer1.ShowNextPage()
```

Do this for the other three buttons, including the appropriate method for each. Compile the project and test these buttons.

In addition to page navigation, you also have the ability to choose the zoom factor that is applied to your report. By default, the zoom is set to 100% of the report size unless you specify otherwise.

In our custom viewer, you should have a drop-down list for the zoom factor. To create our own zoom factor functionality, drag a drop-down list onto the form. Open the properties for your drop-down list (in our example, we have named the drop-down list `ZoomList`).

In the properties for your drop-down list, locate and open the `Items` property, which should open the dialog shown here:

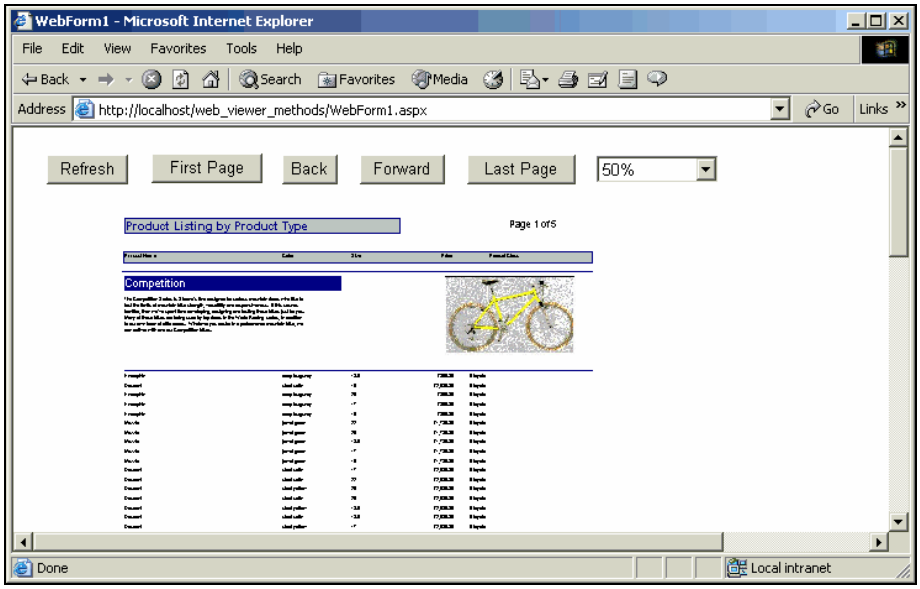


Using this dialog, we are going to create the items that will appear in our drop-down list and specify the corresponding values that will be passed to the form when an item is selected. Use the Add button to add items and make sure that the values correspond to the text you have entered (for instance, Full Size = 100, 50% = 50, and so on).

Once you have entered all of the values, click OK to accept these changes and return to your form's design. To use the `Zoom` method, double-click your drop-down box and add the following code:

```
CrystalReportViewer1.Zoom(DropDownList1.SelectedItem.Value)
```

This is simply calling the `Zoom` method using the item the user selects from your drop-down box. When you run your application and preview your custom viewer, you should be able to select your own zoom factor, and have it appear in the browser by pressing the Refresh button, as shown here:



Searching within a Report

Another powerful navigation feature can be found in the `SearchForText` method within Crystal Reports.NET, which will allow you to search for a specific string that appears in your report.

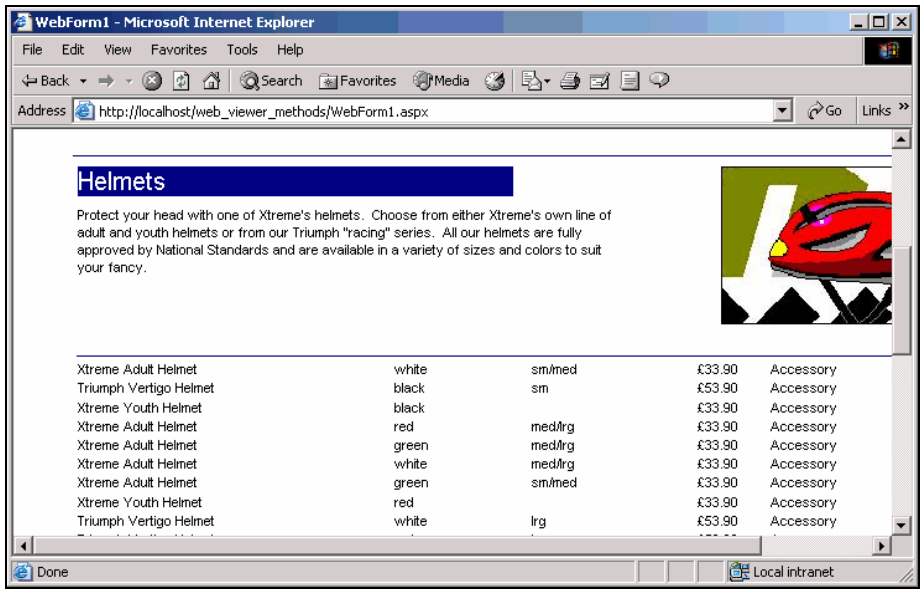
On our custom viewer, we are going to create a textbox and a button labeled Search. We are going to use this textbox to enter some search string and when the user clicks the Search button, we are going to use the `SearchForText` method to find the search string within our report.

To start, we will call our textbox `TextBox_SearchString` and our Search button `Search_Button`. Add these to the design view of our Web Form, remembering to replace the Text property for the button with `Search`.

To use the `SearchForText` method, double-click the Search button and add the following code behind:

```
Private Sub Search_Button_Click(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles Search_Button.Click
  If TextBox_SearchString.Text <> "" Then
      CrystalReportViewer1.SearchForText(TextBox_SearchString.Text, _
      CrystalDecisions.[Shared].SearchDirection.Forward)
  End If
  TextBox_SearchString.Text = " "
End Sub
```

The Crystal Report Viewer will search the entire report and when the value is found, go directly to the page on which it appears (the last line of code above is just to clear the textbox for the next search) This method can be called repeatedly to find all of the occurrences of a particular string – each time it finds the string in your report (in our example below, we searched for `Youth Helmet`), it will jump to that page, as shown below:

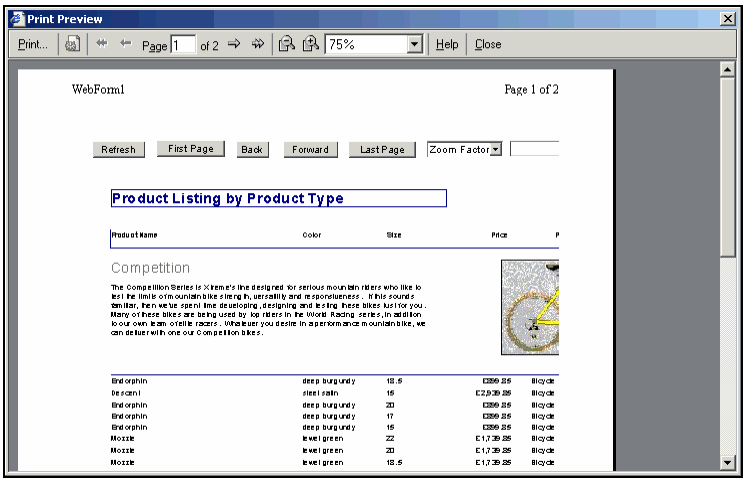


You may have noticed that this method is slightly different between the Windows and web viewers for Crystal Reports: with both types of Forms Viewer, you can pass the additional parameter of `SearchDirection` for searching forwards or backwards through your report. However, in addition, this method in Windows will highlight the found value. The web version does not have this capability.

Printing Your Report

Now, if you have already done some report integration with Windows applications, you may have noticed that the Web Forms Viewer is missing one very important icon – the Print button. When a Crystal Report is viewed on the web, it is actually rendered in static HTML 3.2 or HTML 4.0, with all of the report elements represented in HTML code and syntax.

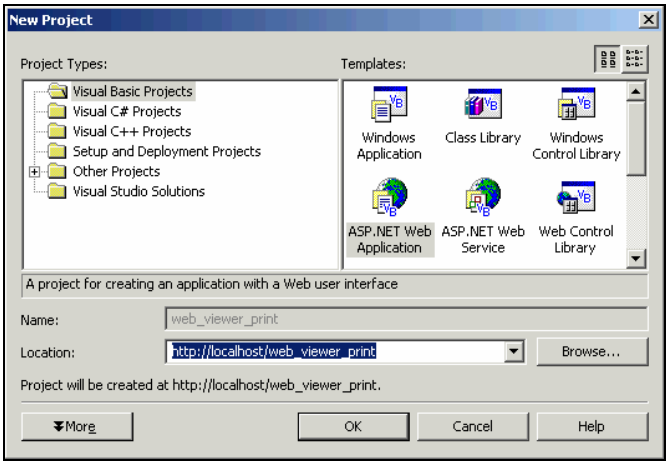
This makes things difficult when it comes time to print your report – in a case where you were just using the plain old viewer with little or no modification, imagine if you were to click on the print button from your browser to print your report. Here is a preview of what your printed report would look like:



It is not very pretty to say the least – if your application uses single-page reports or discrete parts of a report, you may be happy with this, but for the rest of us, there has to be a better solution. So in answer to this limitation in HTML and the way reports are presented in a browser window, we have to come up with some creative solutions to actually print our report to a printer.

The following sections detail the different ways a report can be printed from the Web, as well as some of the advantages and drawbacks of each method. Since we are looking at new functionality within Crystal Reports.NET, we are going to create a new project specifically for this section.

From within Visual Studio, select File | New | Project and from Visual Basic Applications, select ASP.NET Web Application and call this new application `web_viewer_print`. This application is included with the download code. To use the downloaded version a virtual directory should again be created for it in its downloaded location.



Once you have clicked OK, the development environment will open with the default form that we will be using in the section.

We also need to add a report to work with in this section, so select Project | Add Existing Item. Change the drop-down list to show All Files and specify `*.rpt` for the file name to filter the list to show only the available reports. The `web_printing_report.rpt` file is in the code download path `CrystalReports\Chapter04\web_printing_report.rpt`.

Once you have selected the `web_printing_report.rpt` report, click Open and this report will be added to your project in the Solution Explorer – we will be looking at the different methods for printing this report in the following sections.

Now, simply drag a ReportDocument component onto the form, which should offer you `web_viewer_printing.web_printing_report` as first choice in the drop-down box. Select it and drag a CrystalReportViewer onto the Web Form. Now, to bind the `ReportDocument` component to the viewer, merely enter the following code in the Web Form's `Page_Init` event, as we have done more than once in this chapter:

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

Compile and run the application to check that everything is working. We are now ready to start looking at printing this report.

Printing from the Browser

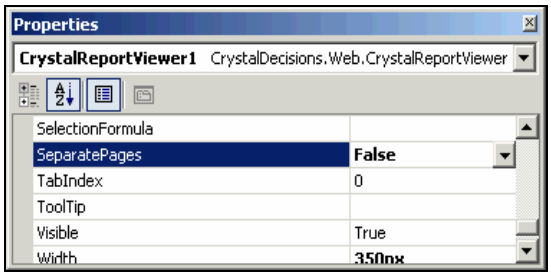
The simplest method for printing a report is to print directly from the browser. You have already seen how well this works, but there are some tricks that we can use to improve the way the report prints if we are forced to use this method.

First of all, you can disable the DisplayGroupTree property if the report is likely to be printed. Do this by setting it to False in the Properties window, or you could do this programmatically by inserting the following code into the `Page_Load` event:

```
CrystalReportViewer1.DisplayGroupTree = False
```

The viewer object model provides a property called `SeperatePages` that by default is set to `True`, meaning that the report is chunked up into individual HTML pages based on the report pagination.

When this property is set to `False`, the report itself becomes one long page, which can then be printed just like any other web page. You can set this property through the property page of the Crystal Report Viewer, as shown here:



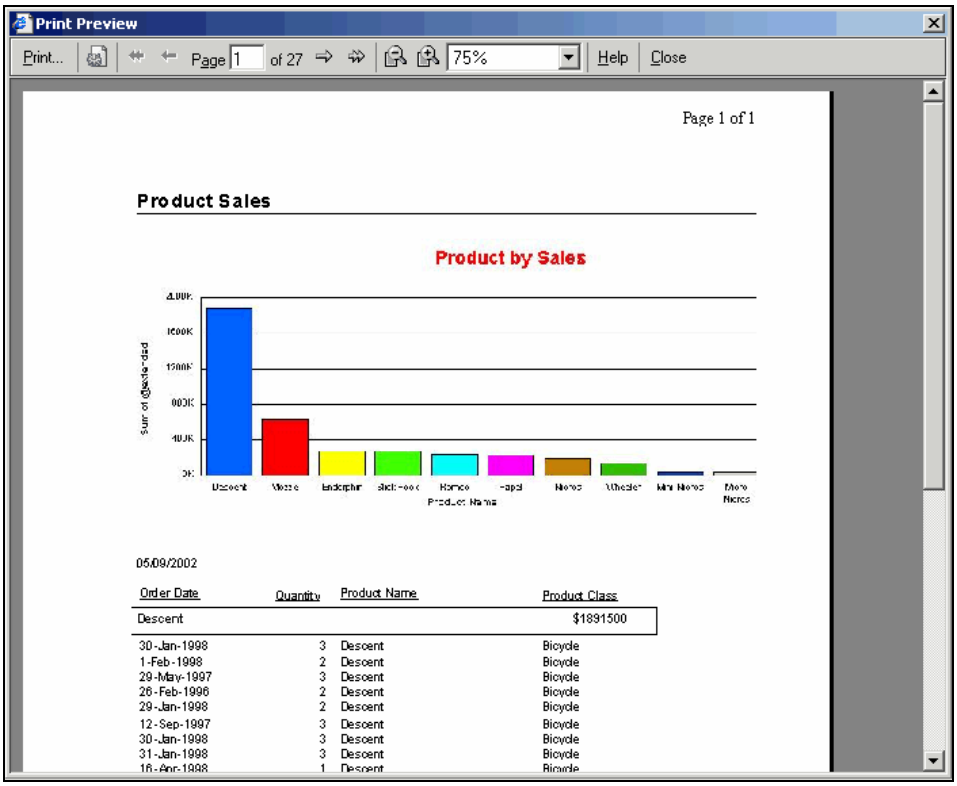
or you can also set this option programmatically:

```
CrystalReportViewer1.SeparatePages = False
```

Another trick is to actually turn off the toolbar and all of the icons so that the output on the page is close to what you would like to see when the report is printed.

```
CrystalReportViewer1.DisplayToolbar = False
```

So with the toolbar turned off and our report showing as one long page, you can then print your report and have a somewhat-decent output as shown here in a preview from Internet Explorer:



The only problem is that this method does not take advantage of any of the neat formatting features for page headers and footers, as the browser just thinks this is one big page to be printed. In addition, the column headings are only printed on the first page, so it is difficult to read the report as you move through the pages.

> **Note: This method is only recommended for reports with a small number of pages (1-20) as the entire report is concatenated into one long page, which may take a while to render on screen or print.**

However, with that said, printing from the browser is the easiest method of printing your report from the web, even with its limitations. For report developers who have put a lot of time and effort into their report design and want that report to be seen and printed by the users (and look good!) we need to look at another solution.

Printing from the Adobe Acrobat Plug-In

Crystal Reports.NET supports many export formats, and one of the more popular ones is Adobe's Portable Document Format or PDF. Using the export functionality within Crystal Reports.NET and a copy of Adobe Acrobat Reader (or the plug-in) installed on the client machine, reports can be printed from the web.

This is one of the methods recommended by Crystal Decisions for printing your reports in a presentation-quality format, and it actually developed the workaround used in this section to help developers who were used to the way Crystal Reports normally operates and were frustrated by not having that print button.

The first thing we need to do is create a new Web form that will contain our instructions. We will call this form `AcrobatPrinter.aspx`, and create it by right-clicking on the project name, and selecting Add | Add New Item. We will then select Web Form and name it as above. Right-click on it and select Set as Start Page.

Draw or drag a button onto the Web form and call it `PDF_Button`, and label it Export via PDF.

Now we need to do some setup to utilize the Crystal Reports Engine (covered in Chapter 8) and set some options available from the `CrystalDecisions.Shared` namespace.

So, we are going to put some code behind our export button to dimension variables for the export options that we want to use, and also for the specific options for exporting to a disk file. Click on the button in the designer, and insert the following code:

```
Private Sub PDF_Button_Click(ByVal sender As System.Object, ByVal e As _
    System.EventArgs) Handles PDF_Button.Click
  Dim myExportOptions As CrystalDecisions.Shared.ExportOptions
  Dim myDiskFileDestinationOptions As _
      CrystalDecisions.Shared.DiskFileDestinationOptions
```

Next, we are going to create a variable to hold the name of the file that we are going to be exporting to, as well as creating a new instance of a Sales Report that has already been added both to the project and to this form, through the `ReportDocument` component.

```
  Dim myExportFile As String
  Dim myReport As New web_printing_report()
```

For our next order of business, we need to set a temporary location for the output file – this can be anywhere on your server – and we are going to build a unique file name using the session ID from the ASP.NET session and tacking the PDF extension on the end, so the file association will work correctly when we go to view this file in our browser.

```
    myExportFile = "C:\CrystalReports\Chapter04\PDF " & _
        Session.SessionID.ToString & ".pdf"
```

Now, for the meat of the matter – actually setting the destination options to export your report to a PDF file and write it to the disk.

```
    myDiskFileDestinationOptions = New
        CrystalDecisions.Shared.DiskFileDestinationOptions()
    myDiskFileDestinationOptions.DiskFileName = myExportFile
    myExportOptions = myReport.ExportOptions
      With myExportOptions
        .DestinationOptions = myDiskFileDestinationOptions
        .ExportDestinationType = .ExportDestinationType.DiskFile
        .ExportFormatType = .ExportFormatType.PortableDocFormat
      End With
```

Then, we call the `Export` method to export our report:

```
    myReport.Export()
```

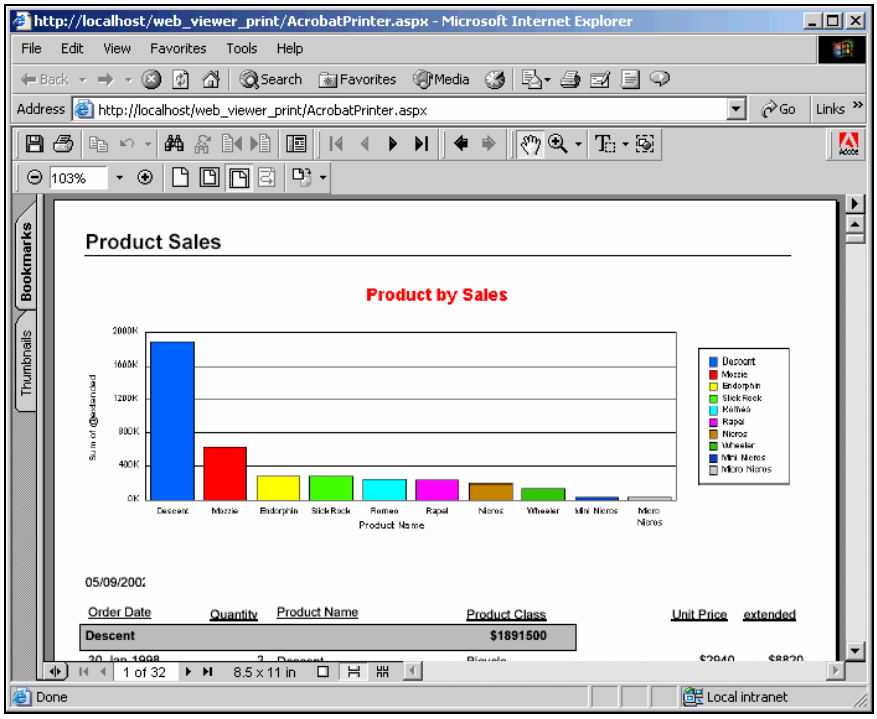
But we are not done yet! We need to take the exported PDF file that has been generated and output it to the browser so the user can view it using the Acrobat Plug-In or standalone viewer. To do that, we are going to use some simple response statements to return the file to the browser:

```
    Response.ClearContent()
    Response.ClearHeaders()
    Response.ContentType = "application/pdf"
    Response.WriteFile(myExportFile)
    Response.Flush()
    Response.Close()
```

Finally, once we have delivered the file to the user, we need to clean up after ourselves and remove the file from the server altogether.

```
    System.IO.File.Delete(myExportFile)
End Sub
```

So when all of this code is put together behind our export button and our application is run, the user can click the button and preview and print the report from Adobe Acrobat, with the page numbering and other features in place, as shown here:



Printing from other Export Formats

In addition to Adobe Acrobat format, you can also print to other supported export formats such as Excel, Word, or others, by changing the file extension, the MIME type, and the `ExportFormatType` property in the code above. There are a number of different destinations that are supported, including:

Name	Description	MIME Type
`Excel`	To export to a Microsoft Excel file	`application/vnd.ms-xls`
`HTML32`	To export to an HTML file compatible with HTML v3.2	`application/html`
`HTML40`	To export to an HTML file compatible with HTML v4.0	`application/html`
`PortableDocFormat`	To export to PDF (Acrobat) format	`application/pdf`
`RichText`	To export to an RTF file for use with Microsoft Word, WordPerfect, and so on	`application/rtf`
`WordForWindows`	To export to a Microsoft Word file	`application/msword`

If you want to export to Word, the RTF export actually provides a better export format. To open the RTF on the client side using Word (instead of the application associated with the RTF file extension), leave the `ExportFormatType` property set to `RichText` but change the MIME type to be `application/msword`.

Using Viewer Events

Viewer events provide the ability to track when different events are fired from the browser – for instance, when the user navigates through the pages of the report or when they refresh the report. These events can then be used to fire other code within your application.

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

In the following section, we will be looking at all of the available events associated with the viewer and their common use – if you would like to try out some of the events listed below, open the custom viewer we were working with earlier in the chapter (`WebForm1.aspx` from the project `web_viewer_properties`) and add a label to your form (call it `Event_Label`) – we'll use this label to notify the user when an event is fired. Clear its Text property. Now we are ready to begin.

Page Navigation Events

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

Property	Description
`CurrentPageNumber`	Returns the current page number
`NewPageNumber`	Gets or sets the new page number

In the example below, the `Navigate` event would fire when a user changed the page within the viewer, resulting in a label that would show the page they are coming from and the page they are navigating to.

Insert the following subroutine into your Web Form code:

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

  If MyEvent.NewPageNumber <> 1 Then
    Event_Label.Text = "Current page: " & MyEvent.CurrentPageNumber & _
    " New Page: " & MyEvent.NewPageNumber
  End If
End Sub
```

So, as the user navigates through the pages, this information is shown and can be used in your application. Compile and run this code to see this happen.

Refresh Events

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

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

Insert this subroutine into your Web Form code, in the same way as we did above. Compile and run. The message should now appear in the label when you hit Refresh.

Search Events

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

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

So by using these event arguments, you could keep a record of what values users searched for or offer a "Top Ten" search facility to let them search using the ten most requested search strings. An example of getting the text that is being used in the search is included below – insert this subroutine into your code, build and run it:

```
Private Sub CrystalReportViewer1_Search(ByVal source As Object, ByVal
    MyEvent As CrystalDecisions.Web.SearchEventArgs) Handles
    CrystalReportViewer1.Search
  Event_Label.Text = "You searched for " & MyEvent.TextToSearch
End Sub
```

Zoom Events

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

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

```
    Select Case MyEvent.NewZoomFactor
      Case "25"
        Event_Label.Text = "You have selected 25%"
      Case "50"
        Event_Label.Text = "You have selected 50%"
      Case "100"
        Event_Label.Text = "You have selected full size"
    End Select
  End Sub
```

Note: For further customization of your report and control of your report's features and functionality, you may want to turn to Chapter 8 to learn how to work with the Crystal Reports Engine, which provides control over your report at run time.

Summary

By now you know how to integrate reporting into both your Windows and web applications, with this chapter focusing on the latter. You should be able to pick the right object model for the functionality you want to provide to your users, as well as work with all of the properties, methods, and events contained within those models.

For our next trick, we are going to look at extending Crystal Reports through the use of XML Report Web Services, which is the topic of Chapter 5.