CHAPTER 28

Crystal Reports with Visual Studio .NET

Although most agree that it got off to a slower-than-anticipated start, Microsoft's Visual Studio .NET development environment appears to be gaining ground as the current state-of-the art in Microsoft-oriented development tools. Organizations find its true object orientation, rich integrated development environment, and from-the-ground-up language independence (among other things) to be worth its disk-space requirements and required effort to learn a new environment.

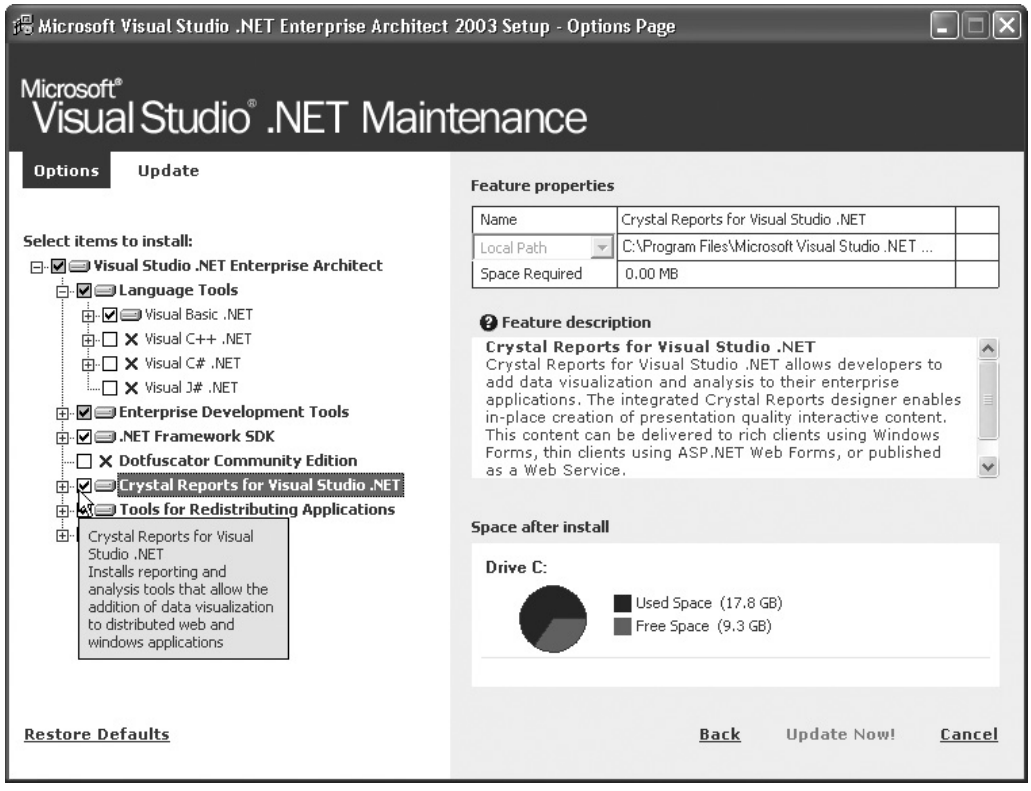
Crystal Reports has played a prominent role in Microsoft development environments in the recent past. Crystal Reports has been bundled with previous versions of Microsoft Visual Studio products (perhaps the sole exception being the concentration on the Microsoft Report Designer included with Visual Studio 6—Crystal Reports was *still* available separately on the program CD). With Visual Studio .NET (VS.NET), Crystal Reports has once again taken on a prominent role, being an integral part of the VS.NET package right "out of the box."

NOTE *While this chapter covers VS.NET usage in Windows application creation, one of the major benefits of VS.NET is the similarity between web and Windows development environments, including Crystal Reports integration. For this reason, you'll find that this chapter is very similar to the last portion of Chapter 22, which concentrates on Crystal web development.*

Crystal/VS.NET Bundle Options

First, you should distinguish the Crystal Reports features bundled with the core Microsoft VS.NET package from those that are added on by a Crystal Reports 10 purchase. If you purchase a copy of Microsoft Visual Studio .NET 2003, Crystal Reports is "bundled" with the VS.NET package. Simply ensure that it is selected when you initially install the VS.NET package to ensure that Crystal Reports features will be available. If you discover that you didn't initially install Crystal Reports, simply rerun VS.NET setup from the Windows

Control Panel and select the Crystal Reports option. This will install the core Crystal Reports bundle that provides most all of the Crystal Reports capabilities you'll need from within a .NET application.

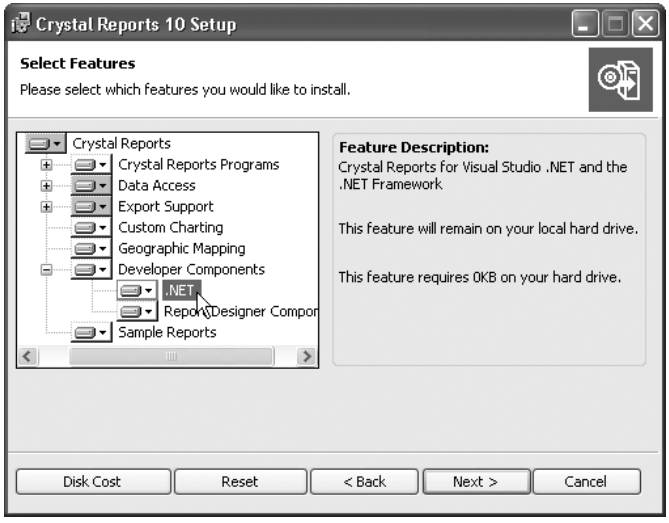


This installs the .NET-bundled version of Crystal Reports, consisting of the Crystal help collection added to the combined VS.NET help collection; and necessary assemblies for designing a report within the VS.NET IDE, integrating the report with a web or Windows application, and creating a Crystal Report web service. This *will not* install a stand-alone copy of Crystal Reports that allows you to design reports outside of the VS.NET IDE.

The version of Crystal Reports installed with VS.NET falls in between Crystal Reports versions 8.5 and 9. While this version will create an .RPT file that can be used outside of VS.NET, it cannot be used with Crystal Reports 8.5 or earlier, as it makes use of the new version 9 .RPT file format that is not backward compatible. However, these .RPT files can be opened and manipulated with stand-alone versions of Crystal Reports 9 or later.

The object models exposed by the bundled version of Crystal Reports are fairly full-featured but not equal in flexibility to those supplied with the COM-compliant Report Designer Component with Visual Basic 6 (discussed in Chapter 27). And, some features and procedures with the bundled version require extra coding steps to navigate through more complex sets of objects than are available with the object model installed with Crystal Reports 10 Developer or Advanced Developer Editions.

If you install Crystal Reports 10 Developer or Advanced Developer Editions, you'll find a .NET option within the Developer Components section of the custom installation option. Ensure this is chosen when you install Crystal Reports 10. If you don't initially choose this option, rerun Crystal Reports setup from the program CD or the Windows Control Panel and choose the .NET option within the Developer Components category. You'll also need to rerun Setup and choose this option if you install (or re-install) VS.NET *after* you install Crystal Reports 10. Repairing a Crystal Reports 10 installation after re-installing VS.NET should also work.



NOTE *You may notice an "Add .NET Components" hyperlink from the initial startup screen that appears when you insert the Crystal Reports 10 program CD. Unfortunately, when you click this link, only a message will appear indicating that .NET components are installed from the actual setup program. You'll need to ensure the proper option from the Developer Components section of the setup program is selected to install the extra version 10 .NET components.*

Installing Crystal Reports 10 Developer or Advanced Developer Edition will supplement the existing Crystal Reports assemblies installed with VS.NET with newer versions that contain additional capabilities. While not an exhaustive list, these version 10 enhancements include some simpler object model calls to perform various report integration functions and some changes to the VS.NET Integrated Report Designer. An additional help collection will also be added to the combined VS.NET help collection. You'll notice, in many cases, Crystal-related help options for both "Crystal Reports" and "Crystal Reports 10."

Also, you'll be able to create reports in the stand-alone copy of Crystal Reports 10, including the interactive design/preview process, which will probably be a vast improvement over the VS.NET IDE report designer, which doesn't provide the quick design/preview paradigm that stand-alone Crystal Reports does. You may then merely import your finished .RPT file into VS.NET for integration.

NOTE *Many of the examples in this chapter are taken from a sample VS.NET Windows application that can be downloaded from CrystalBook.com, this book's companion web site. These examples have been developed with Crystal Reports Advanced Developer Edition installed. As such, some object model calls demonstrated require the additional capabilities of Crystal Reports 10. These are noted in the sample code and in the book.*

Windows Forms Viewer

When integrating a Crystal report into a Windows application with VS.NET, you should first grow familiar with the separation between the actual report itself (the .RPT file you either create within the VS.NET IDE or add to it) and the Windows viewer that actually shows the report from within your application. If you are going to include a Crystal report in a Windows form as part of your application, you'll need to pay close attention to these two distinct objects. If, however, you only wish to include a report in an application for the purpose of exporting to a file location, printing to a local printer, or e-mailing as an attachment, then you may not even need to add a viewer that presents the report in a Windows form.

However, since many Windows applications that include a Crystal report will probably need to present the report to a user in a form, you'll immediately need to determine how to present the report from within a standard VS.NET Windows form. For this, VS.NET includes the *Crystal Windows Forms Viewer.* By dropping a Crystal Windows Forms Viewer onto a Windows form, you can immediately display a Crystal report in a form, including all embedded formatting. This viewer also supports all typical Crystal Reports interaction, including page-on-demand viewing, group tree navigation, report group drill-down, on-demand subreport interactivity, and report exporting and printing.

After creating a Windows application (the language you use to create it is irrelevant—Crystal Reports is entirely language independent in VS.NET), add a Windows Forms Viewer to the desired Windows form by dragging it from the Windows Forms section of the Toolbox. The object, labeled CrystalReportsViewer in the toolbar, can then be resized once it appears on the form. You'll also find a set of properties specific to the Windows Forms Viewer in the Properties box. This is shown in Figure 28-1.

However, merely adding a Windows Forms Viewer to a Windows form won't display anything when the application is run. This is because, as you may gather from the discussion earlier in the chapter on the two distinct parts of a VS.NET Crystal Reports application, you must tie or *bind* an actual Crystal Report object to the Windows Forms Viewer so that there's an actual report to view.

NOTE *Crystal Reports can be integrated in VS.NET applications based on Visual Basic, C#, Managed C++, or other languages supported in the VS.NET development environment. This chapter, however, will concentrate on Visual Basic applications and will demonstrate Visual Basic coding techniques. The material assumes you already have a fundamental knowledge of Visual Studio .NET architecture and coding techniques. This portion of the book will not teach you VS.NET fundamentals.*

Windows Forms Viewer Binding Options

There are numerous ways to bind reports to the Windows Forms Viewer. The option you choose depends largely on the way your application is designed; whether your reports are

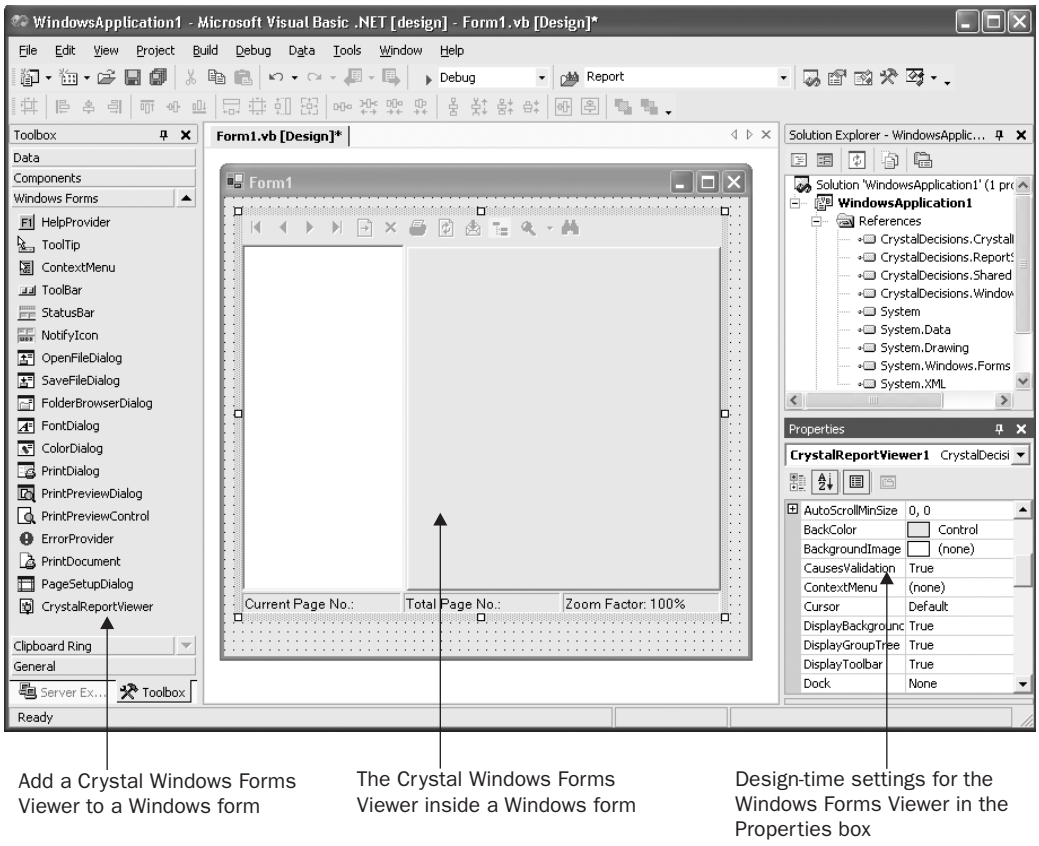


FIGURE 28-1 Crystal Windows Forms Viewer in a Windows form

"external" to your application (located on shared network server at a common file location for use by multiple applications), designed within the VS.NET IDE, contained in a Report Component (discussed later), and so forth.

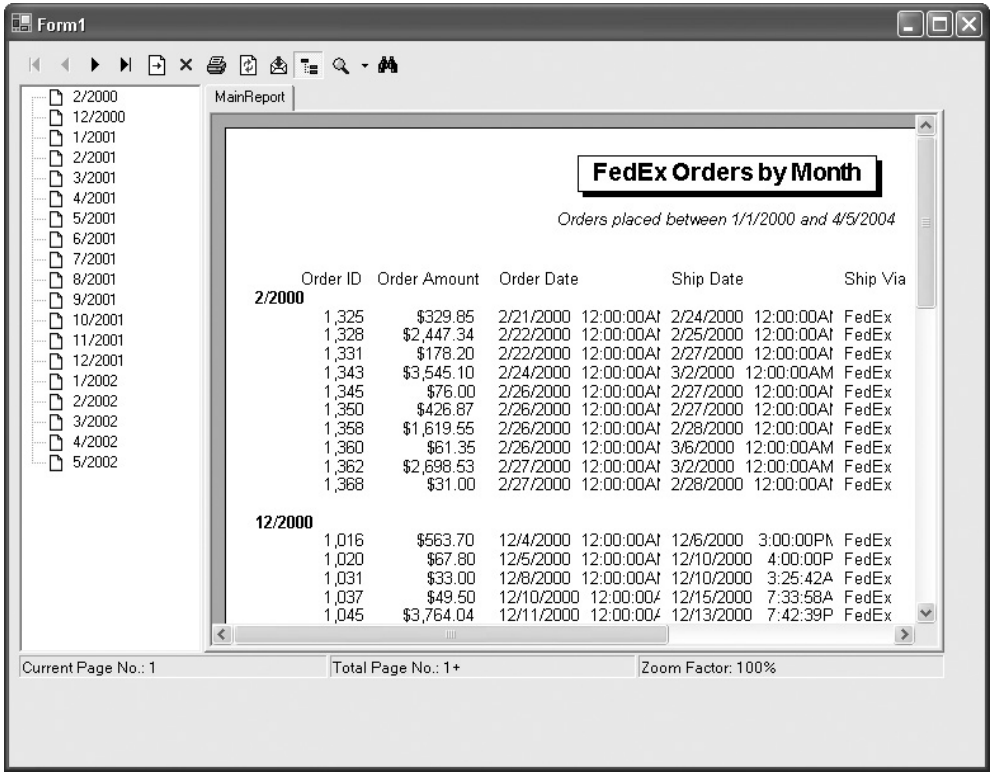
Probably the simplest way to bind a report to the viewer is to specify the full pathname and filename of an .RPT file located on the local or network drive. There are several ways to accomplish this:

- **Specify the viewer's ReportSource property** Make sure the Crystal Windows Forms Viewer is selected in the VS.NET IDE. Navigate to the ReportSource property in the Properties box. Click the down arrow and choose the Browse option. A file open dialog will appear. Navigate through drives and folders until you find the .RPT file you wish to display in the viewer.
- **Set the viewer's ReportSource property in code** The report can be bound at run time by specifying the Crystal Windows Forms Viewer's ReportSource property. For

example, if you display the code window for the Windows form containing the viewer (perhaps the form's Load event), specify the viewer's ReportSource property as follows:

```
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.ReportSource = "C:\Temp\FedEx Orders.rpt"
End Sub
```

When you specify a report source, the Windows Forms Viewer won't show the report at design time. However, when you run the Windows application, the Windows form will display the report inside the Windows Forms Viewer within the form (you may be prompted for database credentials or parameter values first).



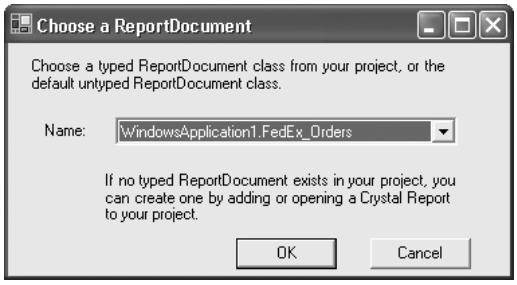
TIP *Two Crystal Windows Forms Viewer properties you'll want to acquaint yourself with are Anchor and Dock. These properties will allow you to "attach" the Windows Form Viewer to its parent Windows form on any combination of sides. If you plan on having the report fill the entire Windows form (this is often the most desirable choice), anchor or dock the Windows Forms Viewer on all four sides. The report viewer will resize inside the Windows form as you resize it.*

You may also want to bind a report you've created within the VS.NET IDE to a report viewer (the Integrated Report Designer is discussed later in the chapter, under "Integrated Report Designer"). Or, you may wish to add an existing .RPT file to your web project and bind it to a viewer. These types of reports that reside in your project are known as *strongly typed* reports, and a class is added to the project for each strongly typed report. In these cases, you'll eventually see an .RPT filename in the Solution Explorer that represents the strongly typed report you want to bind to the viewer. Surprisingly, though, if you attempt to set the viewer's ReportSource property to the .RPT filename, you won't find the file in the drop-down list of available options in the Properties box.

This strongly typed report can be bound using a single line of code in the Windows form's Load event. By setting the viewer's ReportSource property to a new instance of the desired report class (sans the .RPT extension), you can bind the viewer to the report included in the project.

```
'Binds strongly-typed FedEx Orders.RPT file from Solution Explorer
CrystalReportViewer1.ReportSource = New FedEx_Orders
```

You may also wish to bind a report to the Crystal Windows Forms Viewer via strongly typed *report component*. In this situation, you add an additional component to the Windows form from the Toolbox and choose an existing strongly typed report in the project for the component to host. Begin by dragging the ReportDocument from the Components area of the Toolbox to the Windows form. Once you've dropped it on the form, the Choose a ReportDocument dialog box will appear. Click the drop-down list and choose the desired report from the list (you'll see strongly typed reports you've already added to your project).



Once you click OK, the report component will appear in a design time–only area of the Windows form (it will not be visible at run time). You may then merely select the Crystal Windows Forms Viewer in the form and click the drop-down list for the ReportSource property in the Properties box. You'll see all report components you've added to the form in the list. Choose the desired component.

TIP *There are yet additional ways of binding reports to viewers. A complete discussion of all Windows Forms Viewer binding options, including sample code, can be found in VS.NET help. Search for "Report Binding Options for Windows Forms Viewers." There are also samples and examples of report binding that can be downloaded from support.BusinessObjects.com or www.BusinessObjects.com/DevZone.*

Integrated Report Designer

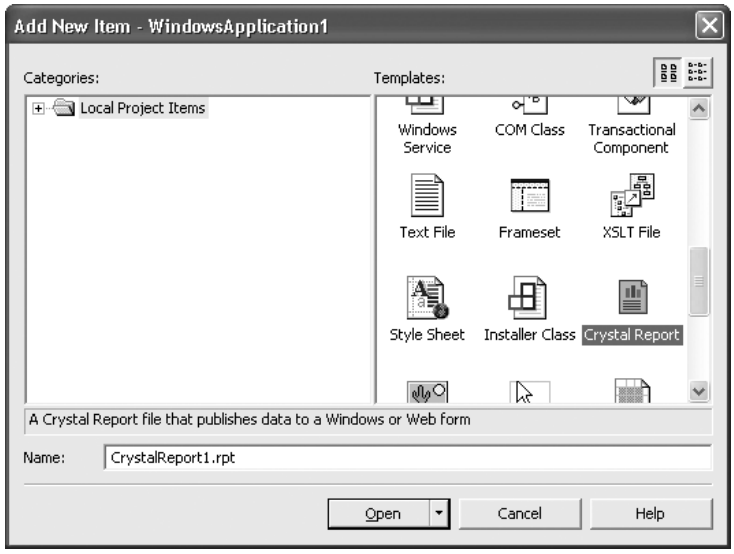
One of the more exciting integration benefits of Crystal Reports and Microsoft development environments is the *Integrated Report Designer.* This feature was originally introduced with the Report Designer Component and Visual Basic 6 (this is covered in Chapter 27). As Microsoft chose Crystal Reports for its integrating reporting tool in Visual Studio .NET, this capability is brought into the VS.NET environment as well. With this design tool built into the VS.NET IDE, you can perform complex report design tasks, such as adding and joining tables, adding fields, grouping records, and formatting objects, much the same way as you can with the full Crystal Reports Designer in an off-the-shelf version of Crystal Reports. When you are finished, the .RPT file you've created or modified will exist within the project as a strongly typed report.

NOTE *One of the initial decisions you'll be making is whether to create reports in the VS.NET IDE or in stand-alone Crystal Reports. While VS.NET allows you to create reports without buying a stand-alone copy of Crystal Reports, your report design choices are more limited in the Integrated Report Designer. For example, you can't simply click a Preview button to see what your finished report looks like—you must place the report in a viewer and run the application every time you want to see any design changes. This, and other differences, may lead you to actually do most of your report design in a stand-alone copy of Crystal Reports and then add the .RPT file to your project with Add | Add Existing Item.*

If you've already worked on another report (perhaps with the full stand-alone copy of Crystal Reports or another VS.NET application), you can add it to your existing project. Simply right-click on the project name in the Solution Explorer and choose Add | Add Existing Item from the pop-up menu. An open file dialog box will appear. Change the file type in the drop-down list to Crystal Report—only .RPT files will then appear in the dialog box. Navigate to the location for your desired report and open it. It will be added to the Solution Explorer. To edit the report, simply double-click it in the Solution Explorer window. The report will appear in its own tab in the VS.NET IDE inside the Integrated Report Designer.

NOTE *Adding an existing report to your project will make a copy of the report in your project folder. Any modifications you make to this report in the Integrated Report Designer won't affect the original report you chose when you added the item.*

If you wish to create a new report from scratch for design in the Integrated Report Designer, begin by selecting the project name in the Solution Explorer. Right-click and choose Add | Add New Item from the pop-up menu. The Add New Item dialog box will appear. If necessary, scroll through the available items until you find the Crystal Report icon. Once you select it, a default filename will appear at the bottom of the dialog box. Either accept it or type the desired filename in before clicking Open.



A new tab will appear in the VS.NET IDE for the new Crystal report, and the Crystal Report Gallery will appear, much as it does when you create a new report in stand-alone Crystal Reports. The Report Wizard and As a Blank Report radio buttons mimic their stand-alone Crystal Reports counterparts. However, the additional From an Existing Report choice appears only in the VS.NET Integrated Report Designer.

Choosing one of the report wizards will display the step-by-step report design dialogs discussed briefly in Chapter 1 of this book. If you choose As a Blank Report, an empty report canvas will appear in VS.NET (the Database Expert won't appear automatically, as it does when using the Blank Report option in stand-alone Crystal Reports). And, if you choose the From an Existing Report choice, an open file dialog will appear where you can navigate to an existing .RPT file to import into the VS.NET Integrated Report Designer.

Once you've made your choices (and, in the case of a report wizard, proceeded through the necessary steps), your report will appear in the Integrated Report Designer inside the VS.NET IDE, as shown in Figure 28-2.

Choosing a Datasource

If you are creating a new report from scratch in the Integrated Report Designer, you'll initially be presented with a blank design screen when you first create the report. As with most report design scenarios, the first step will be to choose a datasource to base your report on.

As with stand-alone Crystal Reports, this is done with the Database Expert. However, unlike stand-alone Crystal Reports, the Database Expert doesn't automatically appear

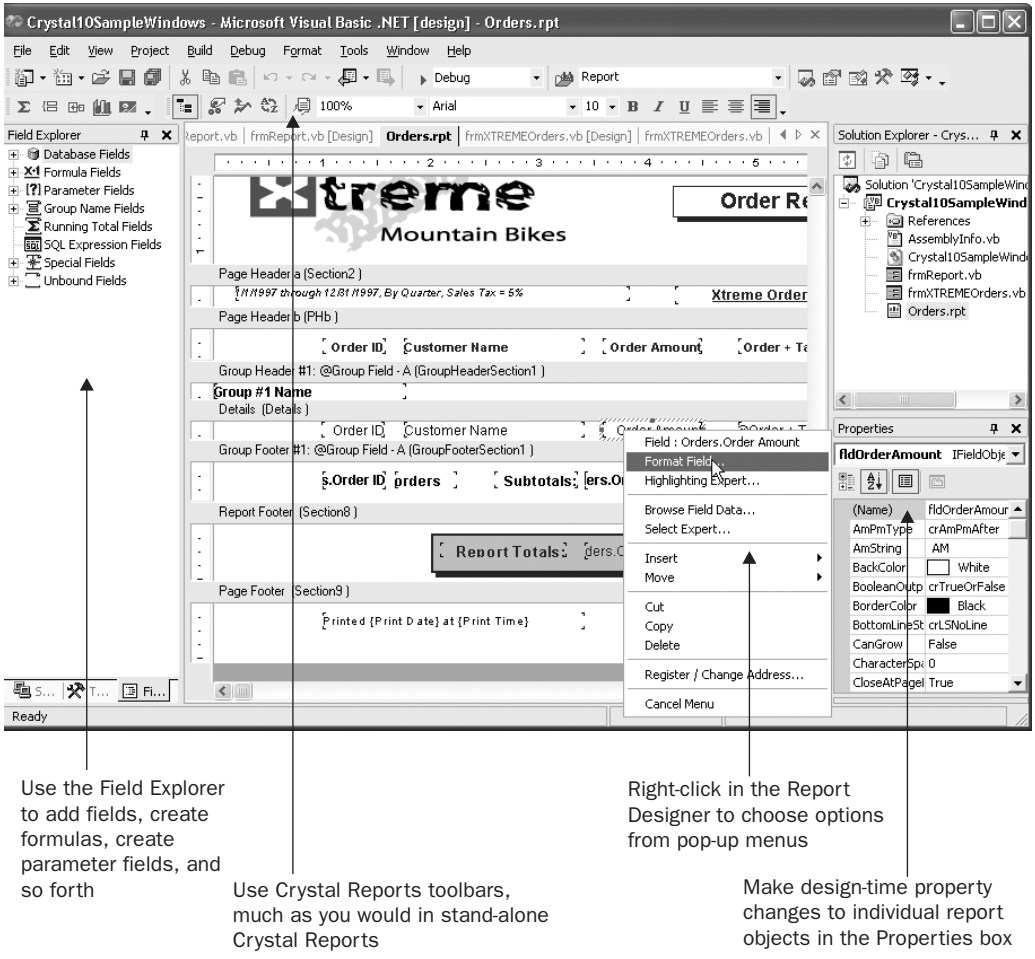
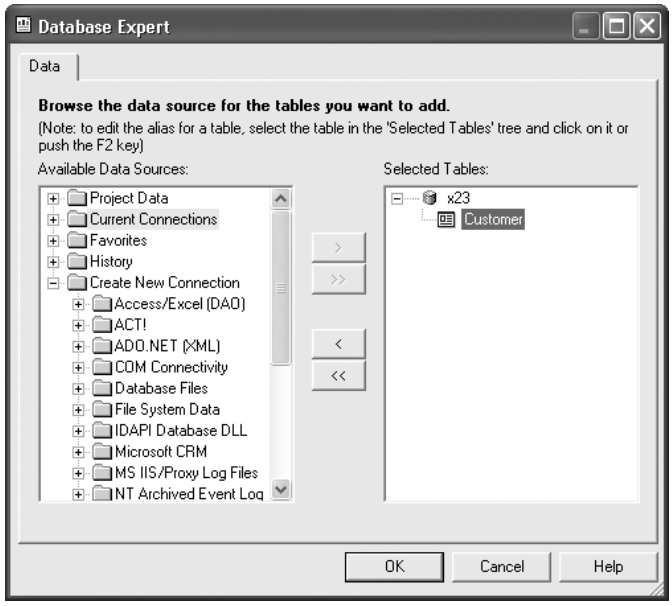


FIGURE 28-2 Integrated Report Designer in VS.NET IDE

when you first create a new report—you need to display it by using a pop-up menu option. Ensure no existing report objects are selected and right-click in the report design area. Choose Database | Database Expert from the pop-up menu. The Database Expert will appear.



Choose the desired datasource category from the list. Note that the choice of categories (and choice of available datasource types) may vary, depending on whether you've installed the Crystal Reports 10 Developer or Advanced Developer Edition—additional datasources are made available when you do this. Also note that some datasources available in the stand-alone version of Crystal Reports, such as SQL Commands or Business Views stored in the Crystal Enterprise repository, won't be available in VS.NET. You will find a Project Data category, however, that will allow you to connect a report to an existing "project" datasource, such as an ADO.NET dataset, that has already been defined in your project.

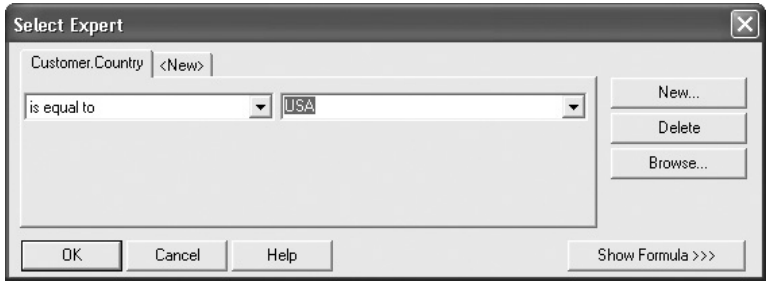
Choose the datasource (or datasources) that you wish to use for your report. You may be required to supply logon credentials to a server to make a connection. Expand the various datasource categories once you've connected to choose individual tables; views; stored procedures; and so forth, that you wish to use in your report. Add each desired table, view, or stored procedure to the Selected Tables list.

If you add more than one item to the Selected Tables list, the Links tab will appear in the Database Expert. Click it and link tables, as described in Chapter 16. Once you've correctly added and linked tables, close the Database Expert by clicking OK.

Selecting Records and Adding Field Objects

Once you've added and linked tables or other datasources, you're ready to add field objects to the actual report canvas. You should also add record selection criteria to ensure that your report will be limited to a meaningful set of data records.

To limit the report to a certain set of records, use the Select Expert or the record selection formula. The Select Expert is displayed by right-clicking on a blank area of the report design canvas and choosing Report | Select Expert from the pop-up menu. You can also click the Select Expert toolbar button in the Crystal Reports toolbar within the VS.NET IDE. If you prefer to edit the record selection formula using the Crystal formula language directly, choose Report | Selection Formulas | Record from the same pop-up menu as the Select Expert. The Formula Editor will appear, where you can create or edit the report's record selection formula. Record Selection is covered in detail in Chapter 8.



Add field objects to your report from the Field Explorer, which is initially docked at the left side of the report design window. These fields can consist of fields directly from your datasource, formulas that you create, parameter fields that prompt the report viewer for a value and then use elsewhere, and so forth. Expand the desired Field Explorer category and drag the desired field to the appropriate report section. You may also right-click a Field Explorer category, or an individual field, and choose options from the pop-up menu.

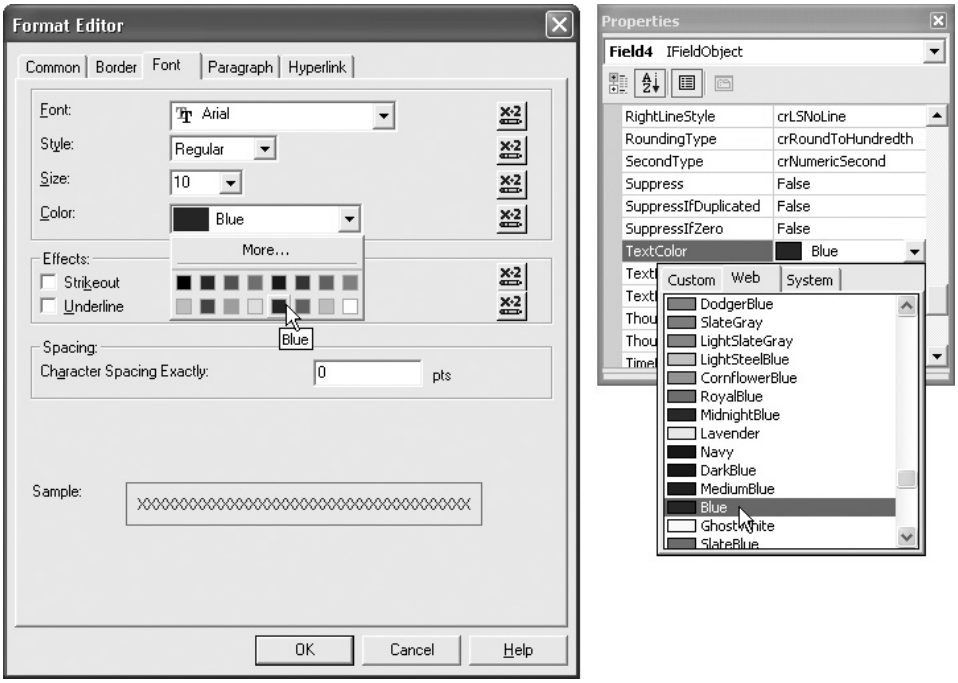
You can use the Field Explorer to create report formulas, parameter fields, and SQL Expressions (all are discussed in various chapters in Part I of this book). You can also drag and drop Special Fields, such as page numbers, print date and time, and so forth, to the report from the Special Fields category. The Unbound Fields category is unique to the VS.NET Integrated Report Designer. Opening this category will reveal a field type for each supported Crystal Reports data type (String, Currency, Date, and so forth). By dragging an unbound field to the report, you are merely adding a "placeholder" object that you'll need to fill with data programmatically at run time with a project datasource, such as an ADO.NET dataset.

TIP *If you plan to modify your report's behavior at run time (almost a given, since you are most probably integrating a Crystal report with a custom application that will control report behavior from your own UI), you should consider using Parameter Fields to control various report behavior. By using Parameter Fields when you design your report to control record selection, report formulas, section or object formatting, and so forth, you can control report behavior at run time by passing different values to the parameter fields from within your VS.NET code. These run-time modification techniques are discussed later in this chapter.*

Formatting Objects and Sections

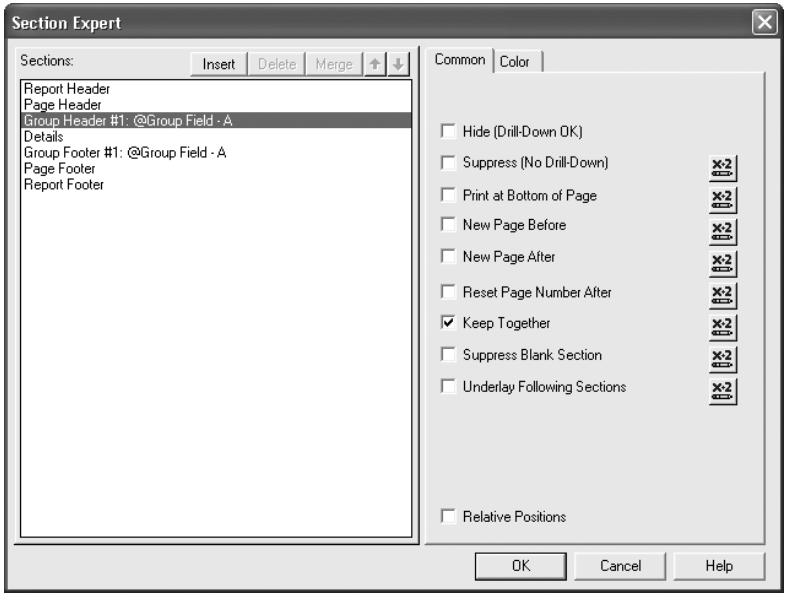
Crystal Reports provides default formatting for individual report objects (fields, formulas, and so forth) and report sections (page header, details, group footer 1, and so forth) when you initially create a report. However, as Crystal Reports is a complete Windows- and web-oriented design tool, you have complete control over a wide variety of properties that affect field and report section appearance.

Format individual report objects by selecting them. Then, make choices from the formatting portion of the Crystal Reports toolbars at the top of the VS.NET IDE. You may also right-click the desired object or objects and choose Format options from the pop-up menu. And, you can also select an object and make choices in the VS.NET Properties box. You'll notice some formatting options change appearance of the report right in the Integrated Report Designer. Other options, however, will only be apparent when you actually run your application and see the report in a report viewer.



Format entire report sections (such as the Details section, Page Header, and so forth) by clicking the gray section title, right-clicking, and choosing options from the pop-up menu. In particular, you can perform all available section formatting by choosing Section Expert from the pop-up menu. This will display the Section Expert dialog box that allows control of suppression, page breaks, section background colors, and much more. Make desired choices and click OK. As

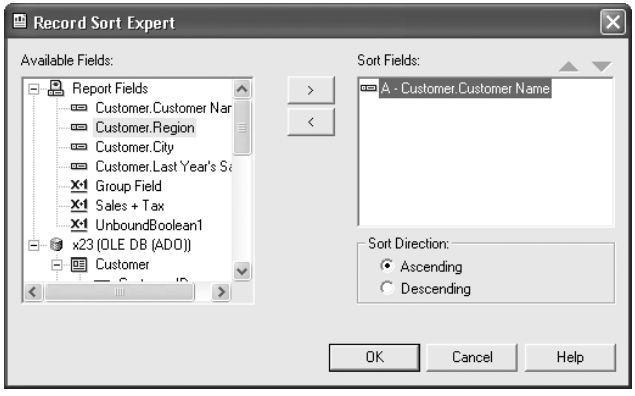
mentioned previously, some of these changes (especially conditional formatting formulas that you may supply) won't be visible until you run your application and see the report in a viewer.



TIP *Formatting report objects is covered in Chapter 9, and section formatting is covered in Chapters 9 and 10. Refer to these chapters for detailed information on report formatting techniques.*

Sorting and Grouping

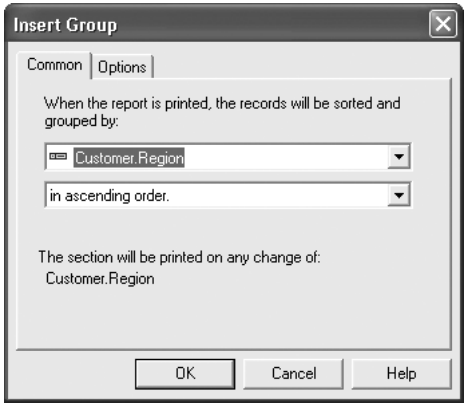
While you can just display data records one after the other in the order the database supplies them, your reports become much more powerful if you at least sort the records in a meaningful fashion by one or more relevant fields. Do this by right-clicking on a blank area of the report designer and choosing Report | Record Sort Expert from the pop-up menu. You can also click the Sort Order button in the Crystal Reports toolbar in the VS.NET IDE. The Sort Order dialog box will appear.



Select a desired sort field from the Available Fields list and click the right arrow to add it to the Sort Fields list. By default, the report will be sorted in ascending (A to Z) order. To change this order, select the desired field in the Sort Fields list and click the Descending radio button at the bottom of the dialog box. The report will be sorted by the chosen field in descending (Z to A) order. You are not limited to sorting the report on a single field—add additional fields in the desired order, choosing ascending or descending sorts for each. If you need to change the priority of fields in the Sort Fields list, simply select the desired field and click the up or down arrow above the Sort Fields list to change the field's priority.

However, you may find that simply sorting the report doesn't provide sufficient flexibility for your needs. In many cases, report grouping is preferable to (or at least helpful in addition to) sorting. By creating report groups, you create "level breaks" that allow placement of summaries and subtotals at the end of each group. Furthermore, these grouping concepts, when used in concert with section formatting (typically hiding of report sections), become the basis for sophisticated drill-down reports that maximize the interactive capabilities of Crystal Reports placed in VS.NET applications.

To create a report group, right-click on a blank area of the report designer and choose Insert | Group from the pop-up menu. You can also click the Insert Group toolbar button in the Crystal toolbars in the VS.NET IDE. The Insert Group dialog box will appear.

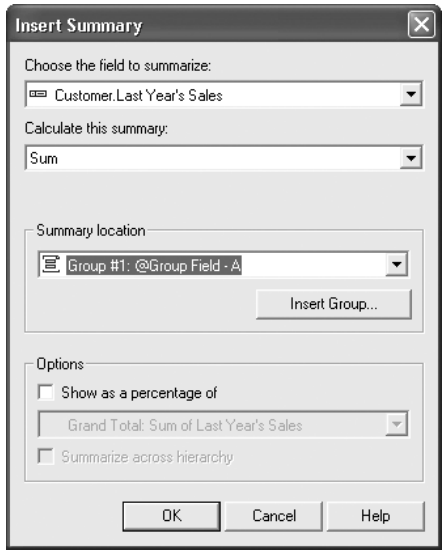


Choose the desired field on which you wish to group the report from the first drop-down list. Then, choose the order in which you want report groups to appear from the second drop-down list. If you wish to set additional group options, click the Options tab of the Insert Group dialog box and make choices. When you're finished, click OK to close the Insert Group dialog box and add the group to the report. You'll notice a group header and group footer section appear in the report for your chosen group.

TIP *Complete details on sorting and grouping are provided in Chapter 3.*

Once you've created necessary report groups, you can insert summary and subtotal fields in the group footers for the groups (as well as grand totals in the Report Header or Report Footer). To insert a summary, you may optionally select the field in the Details section you wish to summarize (although you don't have to). Then, right-click on the field you've selected

and choose Insert | Summary from the pop-up menu. You can also click the Insert Summary toolbar button from the Crystal Reports toolbar in the VS.NET IDE (this button can be used even if you haven't selected a details field first). The Insert Summary dialog box will appear.



If necessary, choose the field you wish to summarize in the first drop-down list (the proper field should already appear in this list if you selected it first). Then, choose the desired type of summary (sum, average, count, and so forth) from the second drop-down list. In the third drop-down list, choose where you want the summary to be placed. Available options will be a grand total (placed in the report footer) or a subtotal placed in the desired group footer. If you wish to use a Percentage Summary Field or summarize across hierarchical groups (described in Chapter 3), check the appropriate boxes. When you click OK, the summary will be placed in the group footer or report footer.

NOTE *While the user interface in the Integrated Report Designer differs somewhat from that in stand-alone Crystal Reports, the general techniques used to design reports are similar. While this chapter covers very general techniques for report design, Part I of this book covers report design techniques in much greater detail. Refer to chapters in Part I for information on various report design steps that also apply to reports designed with the VS.NET Integrated Report Designer.*

VS.NET Report Customization Object Models

While your application may not require any additional capabilities than are provided by a simple report bound to a viewer, there's a good possibility that you'll need to customize one or more aspects of the report at run time from within your application. These customizations can be as simple as supplying the user ID and password for a report's database connection from within your code (so that the user doesn't have to log in to the report's database again,

after they've already logged in to your application). Or, they may require more complex code that changes several report groups, formulas, parameter fields, formatting, or the report's record selection formula all on the fly based on other elements of your application.

Crystal Reports has always been the favorite of developers because of its flexible programming interfaces. And, the versions of Crystal Reports integrated with Visual Studio .NET carry on that tradition by providing rich object models that can control many aspects of report behavior at run time. Installing Crystal Reports 10 Developer or Advanced Developer editions after you've installed VS.NET will improve on this programming flexibility even further.

There are two general ways of customizing your report at run time: using the Crystal Windows Forms Viewer object model or using the Crystal Reports Engine object model. You should look at both object models, as well as your application requirements, to determine which object model you wish to use for run-time customization. Business Objects documentation discourages mixing object model usage—particularly when changing the same properties in both object models. For example, you will probably encounter confusion (and possibly application instability) if you, for example, change a parameter field using the Crystal Reports Engine object model and then change the parameter again using the Crystal Windows Forms Viewer object model before displaying the report in a viewer.

You'll notice that, in particular, the Windows Forms Viewer object model is improved if you install one of the developer editions of Crystal Reports 10. It's quite possible that the viewer object model will provide all the necessary customization you need. However, certain complex reporting requirements that require more involved section formatting, run-time addition or removal of fields, groups, and so forth, will not be satisfied with the viewer object model. These will require use of the Crystal Reports Engine object model that can make significant run-time modifications to a report object *before* it is passed to a report viewer.

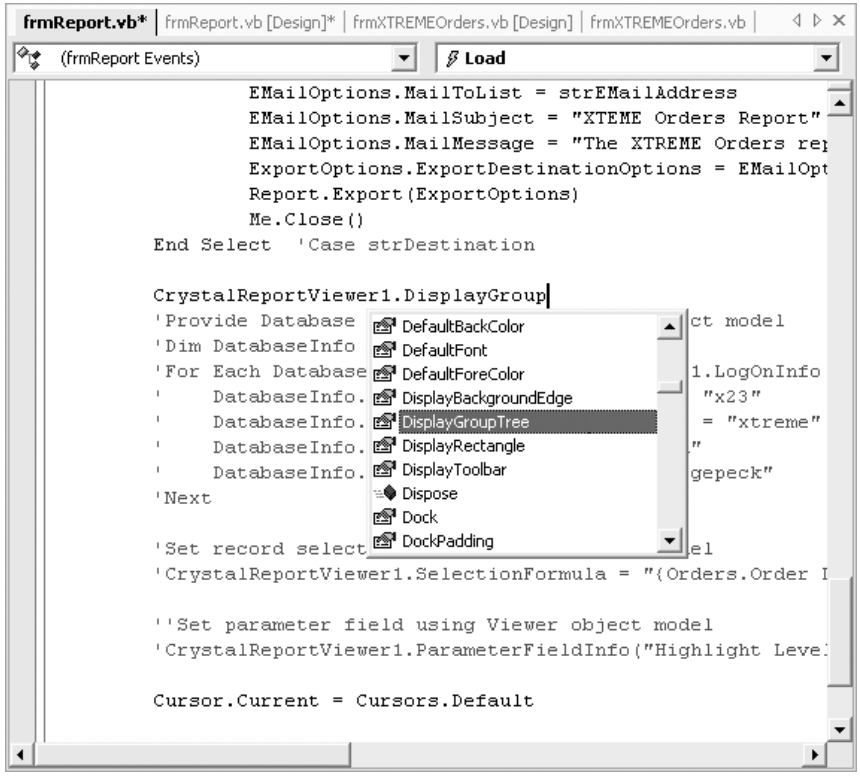
Run-Time Customization with the Crystal Windows Forms Viewer

Some of the most obvious run-time customizations that you may wish to perform with the Crystal Windows Forms Viewer involve basic viewer appearance. You can, for example, control viewer properties at run time that you see in the Properties box at design time (the Anchor property mentioned earlier being a prime example). In addition, commonly used report options, such as the report's record selection formula and parameter field values, can be changed at run time only with calls to the Crystal Windows Forms Viewer object model.

These viewer object model choices are provided by assemblies installed with VS.NET and Crystal Reports 10. In particular, the *CrystalDecisions.Windows.Forms* namespace is added as a reference to your project automatically when you add a Crystal Windows Forms Viewer. This namespace exposes the object model available when using the viewer. When you install Crystal Reports 10 Developer or Advanced Developer edition on an existing computer containing Visual Studio .NET, this namespace is updated with additional version 10 capabilities.

Some initial customization you may wish to perform at run time involves simply viewer behavior, such as visibility of the group tree, appearance of individual toolbar buttons on the viewer toolbar, or complete invisibility of the entire toolbar. VS.NET "intellisense" technology will automatically show you available properties once you type the report

viewer class, followed by a period. You can navigate through the various properties and, if necessary, see available arguments for available properties and methods.



For example, the following code will hide the group tree in the viewer:

```
CrystalReportViewer1.DisplayGroupTree = False
```

And, more involved run-time customization to the viewer can involve instantiating additional objects, setting object properties, and passing the object to a viewer property to change viewer behavior. The following code will supply database credentials to each table object exposed by the Windows Forms Viewer:

```
'Provide Database Credentials via Viewer object model
Dim DatabaseInfo As CrystalDecisions.Shared.TableLogOnInfo
For Each DatabaseInfo In CrystalReportViewer1.LogOnInfo
   DatabaseInfo.ConnectionInfo.ServerName = "x23"
   DatabaseInfo.ConnectionInfo.DatabaseName = "xtreme"
   DatabaseInfo.ConnectionInfo.UserID = "sa"
   DatabaseInfo.ConnectionInfo.Password = "password"
Next
```

Notice the declaration of the TableLogOnInfo class, taken from the *CrystalDecisions.Shared* namespace. This namespace, also added to your project automatically when you add a Crystal Windows Forms Viewer, provides an object model that exposes common objects and properties

that can be used in both Crystal web and Windows applications. For example, this same TableLogOnInfo object could also be applied to the Crystal Web Forms Viewer used in an ASP.NET application, or to objects associated with a report that doesn't even use a viewer.

NOTE *The preceding example shows an object declaration with the fully qualified namespace. If you choose to add an* ***Imports*** *statement to import the namespace, then you would only need to declare the object using its class name without the need to precede it with "CrystalDecisions.Shared".*

The preceding code loops through the individual ConnectionInfo objects within the report viewer's LogonInfo class, setting credentials for each. All tables, regardless of original datasource, as well as subreport connection objects, will be included in the LogonInfo class. Accordingly, you'll need to take extra steps to properly assign values if the source tables in your main report or subreports come from different databases.

Other report properties can also be set with the Crystal Windows Forms Viewer object model. Some of the most obvious include:

- **Specifying database credentials** If the report bound to the viewer is based on a secure database, the viewer will prompt for proper database credentials before displaying the report. If you have already gathered security information elsewhere in your application, you probably won't want your user to be prompted again. You can supply these credentials via the viewer object model, as shown in the previous example.
- **Modifying the record selection formula** This is a very common requirement of report integration applications. You may wish to control report record selection based on some other control on the form or another element of your embedded application logic. This is accomplished by setting the viewer's RecordSelectionFormula property.

```
'Set record selection using Viewer object model
CrystalReportViewer1.SelectionFormula = _
   "{Orders.Order Date} In #" & strStartDate & _
   "# To #" & strEndDate & "#"
```

  Make sure you pass a valid record selection formula, using Crystal Reports Crystal Formula Syntax, to this property. Don't forget to include proper punctuation, such as French braces around field names and apostrophes around string literals. In this example, pound signs are surrounding date data types. You may find discussions in Chapters 5 and 8 helpful in working with record selection formulas.
- **Supplying parameter field values** In many cases, you may actually base your report's record selection on a parameter field. Or, perhaps you have created formulas based on parameter fields that change report grouping or formatting. As such, the ability to pass values to parameter fields at run time is crucial. Here's an example of passing a single value to a single-value (or "discrete") numeric parameter field:

```
'Set parameter via Viewer object model
Dim Parameters As New ParameterFields
Dim Parameter As New ParameterField
Dim ParamValue As New ParameterDiscreteValue

Parameter.ParameterFieldName = "Highlight Level"
ParamValue.Value = sngHighlight
```

```
Parameter.CurrentValues.Add(ParamValue)


Parameters.Add(Parameter)

CrystalReportViewer1.ParameterFieldInfo = Parameters
```

Note that additional objects are instantiated to hold a parameters collection, a parameter field, and a parameter field discrete value. The parameter field name is supplied, using the parameter field's name in the underlying report. The actual discrete value is supplied via a previously defined variable that was properly cast to the appropriate data type. The discrete value object is added to the parameter's CurrentValues collection. The parameter is added to a parameters collection. And, finally, the viewer's ParameterFieldInfo property is set to the parameters collection.

If you chose to simplify this approach with no extra object declarations, the following will perform the same task (note that the underscore is used in the book text only—this line of code must not be separated onto two lines in the code window):

```
CrystalReportViewer1.ParameterFieldInfo("Highlight Level")_
.CurrentValues.AddValue(sngHighlight)
```

Windows Forms Viewer Events

Not only can you set report viewer properties at run time, but you can also trap report viewer events. As Crystal Reports blends into the entire .NET framework, you can trap viewer events in the Crystal Windows Forms Viewer and add code based on them. By selecting the desired Windows Forms Viewer in the Class Name list in the code window, you'll then be able to see a list of events that the viewer fires in the Method Name list.

You may, for example, add code that traps a viewer drill-down event and performs some action based on the group that is drilled into.

```
Private Sub CrystalReportViewer1_Drill(ByVal source As Object, _
     ByVal e As CrystalDecisions.Windows.Forms.DrillEventArgs) _
     Handles CrystalReportViewer1.Drill
  If MsgBox("Are you sure you want to drill into " & _
  e.NewGroupName & "?", MsgBoxStyle.YesNo, "Drill Down Detected...") _
  = MsgBoxResult.No Then
    e.Handled = True
  End If 'MsgBox
End Sub
```

In this example, the event class exposes DrillEventArgs from the CrystalDecisions.Windows .forms namespace, including various properties to indicate what is being drilled into. Here, the code displays a message box asking the user to confirm the drill-down, including display of the group being drilled into via the event's NewGroupName property. If the user provides a response of No, the drill-down is canceled by setting the class's Handled property to True.

Run-Time Customization with the Crystal Reports Engine

So far, this chapter has concentrated entirely on run-time customization with the Crystal Windows Forms Viewer object model (with classes exposed by the CrystalDecisions.Windows .Forms namespace) and the shared object model (with classes exposed by the CrystalDecisions .Shared namespace). There's still yet another object model/namespace that exposes classes you can use to customize report behavior at run time.

By using the *Crystal Reports Engine* object model, you can customize report behavior at run time using techniques similar to those discussed previously in the chapter. However, this object model (exposed by the CrystalDecisions.CrystalReports.Engine namespace) offers properties, methods, and events that are strictly limited to the core report object you are integrating in your application, independent of any viewer you may have added to your application. Also, a much larger number of customization options are available with the Report Engine. In addition to setting parameter field values, supplying database credentials, and setting the report's record selection formula, the Report Engine offers the ability to control object formatting, supply new formula contents, export the report to various file formats, and much more.

Any customization you perform here typically takes place *before* you bind a report to a viewer (or, in situations where you don't even add a viewer to your project). Once you've made these customizations to the report object, they will take effect when the report is bound to the viewer (or, in the case of a non-viewing application, whenever you execute a report method to ultimately output the report to a printer, exported file format, and so forth).

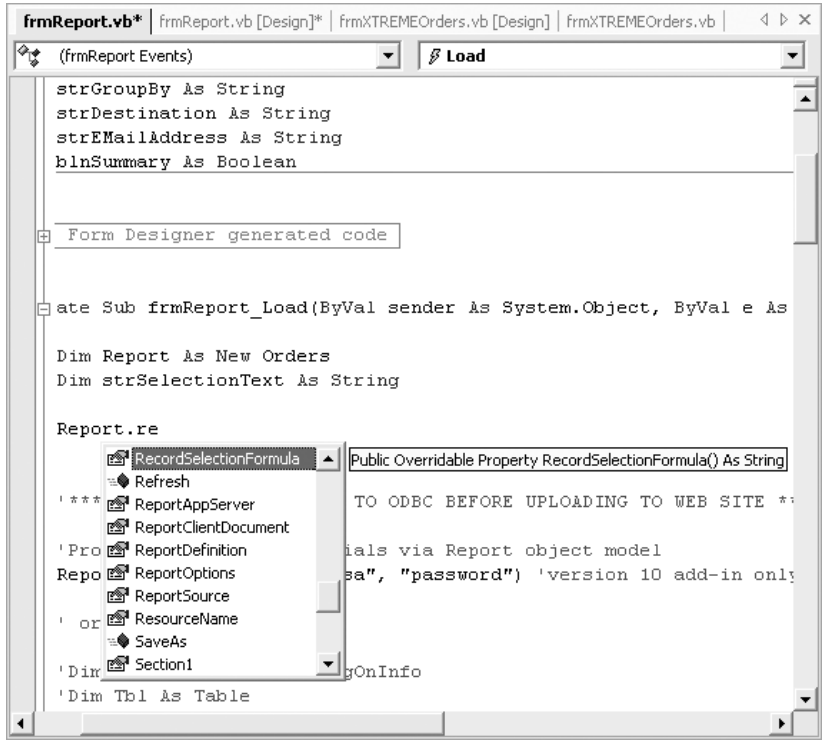
As discussed earlier in the chapter, you'll find properties and methods in the Report Engine that duplicate those found in the viewer object model. For example, you can supply database credentials, set the record selection formula, and supply parameter field values via either object model. The decision as to which object model to use is dependent upon your application particulars and how detailed your integration requirements are. Again, Business Objects recommends that you avoid duplicate calls from different object models in the same application. So, as you develop your application, look at which object model provides the best combination of simplicity and functionality depending on your requirements.

To make use of the Report Engine, you must ensure that the CrystalDecisions .CrystalReports.Engine namespace is being referenced in your project. In most cases, you'll also need references to the CrystalDecisions.Shared namespace. Typically, if you add a Crystal Windows Forms Viewer to the project, these namespaces will automatically be added. If, however, you are designing an application that does not make use of a Windows Forms Viewer, you'll need to use the VS.NET Add Reference option to add these namespaces to your project. Also, you'll need to declare references to several classes from these namespaces in your code and may prefer to use the Imports statement to allow class references without the fully qualified namespace name.

One of the first requirements of using the Report Engine is instantiation of a report object. As discussions earlier in the chapter indicate, there are several ways of adding reports to your project. And, there are several types of reports (un-typed, strongly typed, and so forth) that can be added. As an example, you may have added a strongly typed report to your project, either by using Add | Add Existing Item with an existing .RPT file or by using Add | Add New Item and designing a new Crystal Report. You then must declare an object to refer to the strongly typed report and expose the Report Engine's object model. Here's an example:

```
Dim Report As New Orders
'refers to strongly-typed Orders.rpt in project
```

Once you've declared this object, you will notice many of the Report Engine properties and methods available via VS.NET's "Intellisense" technology.



Some of the more common run-time customizations you can make with the Report Engine (but, by no means, an exhaustive list) include:

- **Supplying database credentials** If your report is based on a secure database, you will probably prefer that your application code provide database credentials to prevent your report viewer from being prompted, or from your application failing if your report is not bound to a viewer. There are several ways for the Report Engine to supply these credentials. Examine the following:

```
'Provide database credentials via Report object model
Report.SetDatabaseLogon("sa", "password")
```

This single line of code (not including the comment) supplies a user ID and password to every datasource in the report, including those contained in subreports. The SetDatabaseLogon method greatly simplifies supplying database credentials if your report is based entirely on a single database connection.

A second permutation of SetDatabaseLogon accepts four arguments:

```
Report.SetDatabaseLogon("sa", "password", "SQLServer1", "xtreme")
```

This allows a database server and a database name to also be specified, in case you've made use of several different connections in your report. You could repeat this line of code for each server used in the main report and subreports.

If you need a more granular way of supplying database credentials for individual table objects in your main report and subreports, you've got a little more coding ahead of you. Examine the following:

```
Dim Logon As New TableLogOnInfo
Dim Tbl As Table
For Each Tbl In Report.Database.Tables
   Logon = Tbl.LogOnInfo
   Logon.ConnectionInfo.ServerName = "SQLServer1"
   Logon.ConnectionInfo.DatabaseName = "xtreme"
   Logon.ConnectionInfo.UserID = "sa"
   Logon.ConnectionInfo.Password = "password"
   Tbl.ApplyLogOnInfo(Logon)
Next Tbl

'Provide database credentials to subreports
Dim Subreport As ReportDocument
For Each Subreport In Report.Subreports
   For Each Tbl In Subreport.Database.Tables
      Logon = Tbl.LogOnInfo
      Logon.ConnectionInfo.ServerName = "SQLServer1"
      Logon.ConnectionInfo.DatabaseName = "xtreme"
      Logon.ConnectionInfo.UserID = "sa"
      Logon.ConnectionInfo.Password = "password"
      Tbl.ApplyLogOnInfo(Logon)
   Next 'Tbl
Next 'Subreport
```

In the preceding code fragment, a TableLogonInfo class (from the CrystalDecisions .Shared namespace) and a Table class (from the CrystalDecisions.CrystalReports

.Engine namespace) are instantiated. Then, each table in the main report is cycled through, supplying database credentials via the ConnectionInfo property.

To accommodate subreports, a ReportDocument class is instantiated (this comes from CrystalDecisions.CrystalReports.Engine) to cycle through the main report's Subreports collection. For each subreport, similar loops through the Database.Tables collection are made to supply database credentials.

While the preceding code will work with Crystal Reports 10, earlier versions (including Crystal Reports included with VS.NET) do not expose a Subreports collection. In this situation, code to deal with subreports must be even more granular, as the following example shows.

```
crSections = Report.ReportDefinition.Sections

For Each crSection In crSections
   crReportObjects = crSection.ReportObjects

   For Each crReportObject In crReportObjects
     If crReportObject.Kind = ReportObjectKind.SubreportObject Then
       crSubreportObject = CType(crReportObject, SubreportObject)
       subrpt = _

         crSubreportObject.OpenSubreport(crSubreportObject.SubreportName)
       For Each subtbl In subReport.Database.Tables
          str &= subReport.Name & "-" & subtbl.Name & " - "
          objTableLogonInfo = subtbl.LogOnInfo
          objTableLogonInfo.ConnectionInfo = objConnInfo
          subtbl.ApplyLogOnInfo(objTableLogonInfo)
          subtbl.Location = _
           subtbl.Location.Substring(subtbl.Location.LastIndexOf(".") + 1)
          str &= subtbl.TestConnectivity() & vbCrLf
        Next
      End If
   Next
Next
```

- **Setting record selection** As with the viewer object model, you can set the report's record selection formula at run time via a simple property setting in the Report Engine. You will probably want to base this on other controls or conditions in your application. For example:

```
Report.RecordSelectionFormula = "{Orders.Order Date} In #" _
     & strStartDate & "# To #" & strEndDate & "#"
```

will return records where the order date is between values defined in two string variables. Ensure that you pass a syntactically correct formula using the Crystal Reports Crystal Syntax, including French braces around field names and apostrophes surrounding string literals (the pounds signs in this example indicate date values). Discussions on record selection and formula syntax can be found in Chapters 5 and 8 in this book.

- **Providing parameter field values** A very common requirement of run-time report modification is dealing with parameter fields. As many report customizations can

be accomplished by designing reports with parameter fields, you may find this one of the most common run-time modifications you'll be required to perform. While the viewer object model can change parameter field values at run time, you may prefer (or be required) to use the Report Engine to supply parameter values at run time. If you have installed Crystal Reports 10, the report object's SetParameterValue method is a simple way to supply a value to a parameter field:

```
'Supply parameter field using Report object model
Report.SetParameterValue("Highlight Level", sngHighlight)
```

The SetParameterValue method demonstrated here accepts two arguments: the parameter field to supply a value to (which can be specified either by name or index), and the value to pass to the parameter field (in this instance, a variable that has been cast elsewhere in the code to match the data type of the parameter field).

The SetParameterValue method is fairly flexible in the data types and parameter field types (such as multi-value parameter fields) that it can accommodate. It can also be used to supply parameter values to subreports. The Crystal Reports 10 help collection contains complete details.

If you have more complex parameter requirements that may not be accommodated by the SetParameterValue method, you may modify parameters via an additional object. Here's an example that sets a simple, single-value parameter field (note that the line continuation character is used in the book only—the code must be on a single line in the code editor):

```
Dim ParamValue As New ParameterDiscreteValue
ParamValue.Value = sngHighlight
Report.ParameterFields("Highlight Level").CurrentValues._
   Add(ParamValue)
```

A discrete parameter class is instantiated to hold a single discrete (single-value) parameter field. The object's Value property is set to a variable (again, properly cast earlier in the code). And, finally, the parameter object is added to the existing parameter field's CurrentValues collection (which, for a single-value parameter field, will contain only one member). The parameter field can be obtained from the report's ParameterFields collection, either by name or by numeric index.

- **Modifying existing formula contents** One area of report customization that can be accomplished only with the Report Engine is run-time report formula modification. For example, you may wish to change some sort of calculation or string customization contained in a report formula to reflect a user control or other element of your custom application. By modifying a report formula at run time, you can change report calculations or customized fields based on your application elements. For example:

```
'Set formula via Report object model
If sngTaxRate = 0 Then
   Report.DataDefinition.FormulaFields("Order + Tax").Text = _
      "{Orders.Order Amount}"
Else
   Report.DataDefinition.FormulaFields("Order + Tax").Text = _
      "{Orders.Order Amount} + {Orders.Order Amount} * " & _
           Trim(CStr(sngTaxRate)) & "/ 100"
End If 'sngTaxRate = 0
```

You may navigate directly to the FormulaFields collection of the report's DataDefinition object. Notice that the particular collection member can be retrieved by name, in addition to numeric index (this is a refreshing change from many collections in the older Report Designer Component, which could only be accessed via numeric index). By setting the formula's Text property, you can change the formula contents at run time. In this example, a variable is examined indicating the tax rate supplied by the user. The formula that calculates the order amount plus tax is modified, depending on what the user supplies as the tax rate. Chapter 5 discusses report formulas in greater detail.

- **Supplying text object text** While there are several ways to display a custom string or "message" on a report (perhaps you wish to display various run-time options chosen by the report viewer on the report itself), one way is to set the actual text displayed by a text object at run time. This capability is provided by the Report Engine only and cannot be accomplished by using any of the viewer object model calls. Here's an example:

```
'Set text object via Report object model
Dim SelectionText As TextObject
'CrystalDecisions.CrystalReports.Engine.TextObject if no Imports
SelectionText = _
   Report.ReportDefinition.ReportObjects("SelectionText")
SelectionText.Text = strSelectionText
```

Instantiate a TextObject object (supplied by the CrystalDecisions.CrystalReports .Engine namespace). Set it to the proper text object in the report by navigating through the report's ReportObjects collection from the ReportDefinition object. Then, set the text object's Text property to change the actual text contained in the text object on the report.

- **Controlling section formatting** The Print Engine object model exposes all the report areas and sections within the source report. You may set formatting options for these sections and areas, such as page control, background color, section suppression and hiding, and so forth. Here's an example:

```
If blnSummary Then
   Report.ReportDefinition.Areas("Area4").AreaFormat._
        EnableHideForDrillDown = True
   Report.PHb.SectionFormat.EnableSuppress = True
End If 'blnSummary
```

A variable is tested to see if the user chose to see a summary report. If so, the Group Header 1 (the fourth report "area") is hidden for drill-down. And, the Page Header B "section" is suppressed. Note the difference between Area and Section objects—an area refers to an entire series of related sections. For example, in the case of a report containing four details sections, the Report Engine will expose a single details "area" object, but individual details a, b, c, and d section objects. Some formatting is common to both areas and sections, while other formatting is specific to one or the other.

Printing Reports to a Printer

Windows applications often require that an integrated Crystal report be printed on a local or network printer. While the Windows Forms Viewer includes a print button that will print a report when clicked, you may wish to exert more control over printing from within your application code. Both the Windows Forms Viewer object model and the Report Engine object model expose methods to print a report to a printer. Depending on your application's requirements, you may make use of either.

The viewer object model exposes a simple PrintReport method that you may execute in code, as follows:

```
'Print using Viewer Object Model
CrystalReportViewer1.PrintReport()
```

This is functionally equivalent to the user clicking the Print button in the Windows Forms Viewer. It will launch a standard Windows Print dialog box, whereby the user can specify printing particulars, such as the page range. Note that you *cannot* change this behavior—the print dialog box will always display and the user will need to specify any report printing particulars at run time.

If you want more granular control over report printing, or your particular application does not make use of the Windows Forms Viewer, use the Report Engine's PrintToPrinter method, as in the following sample:

```
'Print using Report Engine
Report.PrintToPrinter(1, False, 0, 0)
```

PrintToPrinter accepts four arguments: the number of copies to print, whether to print the report in collated form, the starting print page, and the ending print page (setting the print pages to zero will print the entire report).

Furthermore, you can access various properties within the Print Engine report object's PrintOptions class. Class members include such settings as page orientation, page size, margins, and so forth. If you want to gather some of these values from the user, you may design your own custom print options dialog box, or call a standard Windows print options dialog box, and pass resulting values to the report's PrintOptions class.

Exporting or E-Mailing Reports

While the Crystal Windows Forms Viewer includes an Export toolbar button that allows the viewer to export the viewed report to various file formats, you may wish to do this programmatically. For example, the only available destination for reports exported by clicking the export button is disk—you may prefer to e-mail a report instead.

The simplest way to export a report is via the Report Viewer object model. Here's an example:

```
'Export via viewer object model
CrystalReportViewer1.ExportReport()
```

This is functionally equivalent to the user clicking the Export button on the Windows Forms Viewer. A dialog box will be presented to the user requiring them to choose a location and filename for the export, as well as the file format to export to. You cannot control any of these options programmatically.

For more controlled exporting, you'll need to use the Report Engine. Here's an example of more involved exporting that attaches a file to an e-mail message:

```
Dim ExportOptions As New ExportOptions
Dim EMailOptions As New MicrosoftMailDestinationOptions
ExportOptions.ExportFormatType = ExportFormatType.PortableDocFormat
ExportOptions.ExportDestinationType = ExportDestinationType.MicrosoftMail
EMailOptions.MailToList = strEMailAddress
EMailOptions.MailSubject = "XTEME Orders Report"
EMailOptions.MailMessage = _
   "The XTREME Orders report is attached in PDF format"
ExportOptions.ExportDestinationOptions = EMailOptions
Report.Export(ExportOptions)
```

In this example, both ExportOptions and MicrosoftMailDestinationOptions objects are instantiated (both classes are exposed by the CrystalDecisions.Shared namespace). The ExportOptions export format type is set to Adobe Acrobat Portable Document Format (PDF), and the export destination type is set to Microsoft Mail. Various MicrosortMailDestinationOptions are used to set the e-mail's To address, subject, and message body text. The ExportOptions ExportDestinationOptions property is set to the MicrosoftMailDestinationOptions object. And, finally, the report object's Export method is executed, with the ExportOptions object being supplied as an argument.

Using Crystal Reports Web Services

One of the new features of Visual Studio .NET is the web service: a web server–based "application" that exposes a set of data based on the Extensible Markup Language, or XML. The web service exposes its actual data, as well as its data layout (or schema), through the Hypertext Transport Protocol, or HTTP. Because the XML data layout is simply an extension of HTML and because HTTP is such a ubiquitous communications protocol, virtually all networks, intranet systems, and Internet connection methods (dial-up, firewalls, and so forth) work with these standards already.

Crystal Reports for Visual Studio .NET fully supports web services—being able to both create and "consume" web services. Creating a web service based on a Crystal report exposes a report via HTTP to anyone who can consume a web service anywhere on an intranet or the Internet. VS.NET also allows you to create projects that use or "consume" published Crystal Report web services. In this scenario, you create a Windows or web application, including a Windows Forms Viewer or Web Forms Viewer, which is then bound to the report exposed by the web service.

NOTE *Creating web services and consuming web services in VS.NET web applications is covered in Chapter 22.*

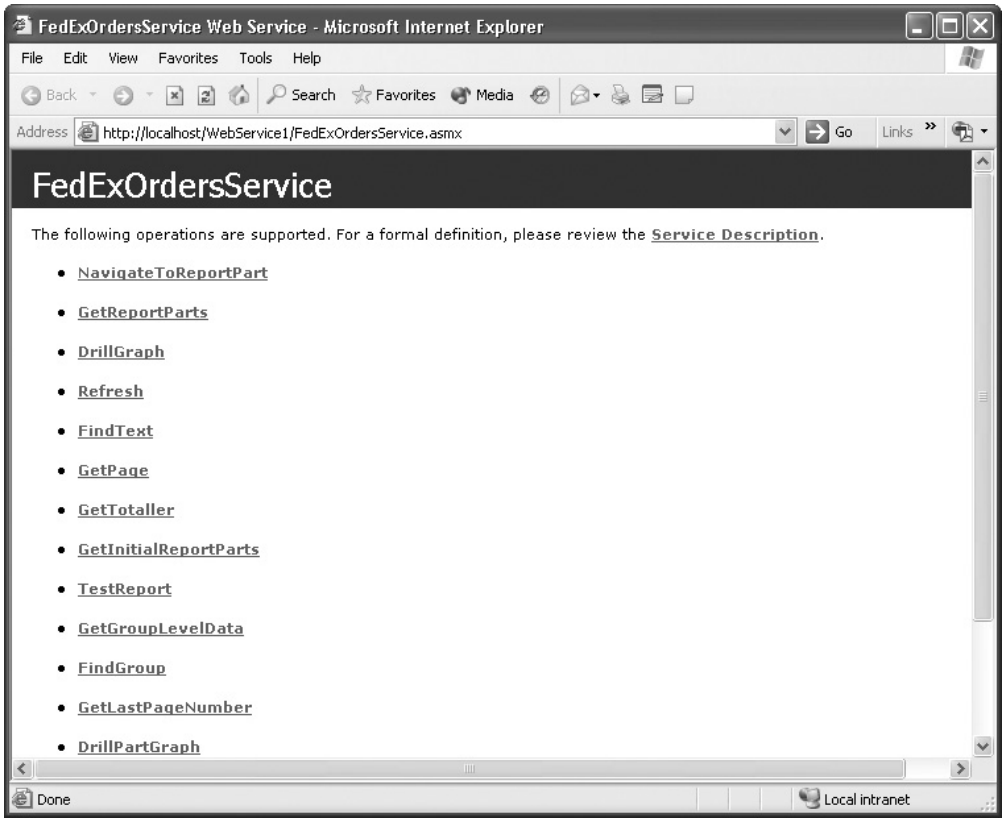
Consuming Web Service Reports in a Windows Application

Once a web service has been created and compiled, you can actually use a web browser to view the properties of the web service. The URL will be provided to you by the web

administrator, or the person who created and "exposed" the web service. Typically, report web service URLs are based on the application or web service used to publish the report, and the report name. For example, if a web service is created with the name of WebService1, and it publishes a report named FedExOrders.rpt, a FedExOrdersService.asmx file will be created in the WebService1 virtual directory. If you type in the following URL in a standard web browser:

```
http://<server name>/WebService1/FedExOrdersService.asmx
```

you'll see the properties exposed by the report web service. You may click various hyperlinks on this page to see the underlying XML that the web report service exposes, but the report will not be viewable in the browser directly using this URL.



In order to view or "consume" a report web service, you must create a VS.NET Windows application and add a Crystal Windows Forms Viewer to it. You must then bind the Windows Forms Viewer to the report exposed by the report web service. As you might

expect, there are several ways to reference a report web service in your Windows application and bind the Windows Forms Viewer to it.

Binding by URL as the ReportSource

Binding report web services to the Windows Forms Viewer must be done in code. The viewer's Report Source property in the Properties box is unable to accept a URL as a value. For this reason, you'll need to bind the report to the viewer's ReportSource property in the code editor (probably in the parent form's Load event). Here's an example:

```
Private Sub Form1_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
    CrystalReportViewer1.ReportSource = _
     "http://localhost/WebService1/FedExOrdersService.asmx"
End Sub
```

This code fragment shows the Windows Forms Viewer being bound to a report web service exposed on "localhost" (your own computer's web server). The exposed report is originally entitled FedExOrders.rpt and is published as a web service from a web project named "WebService1."

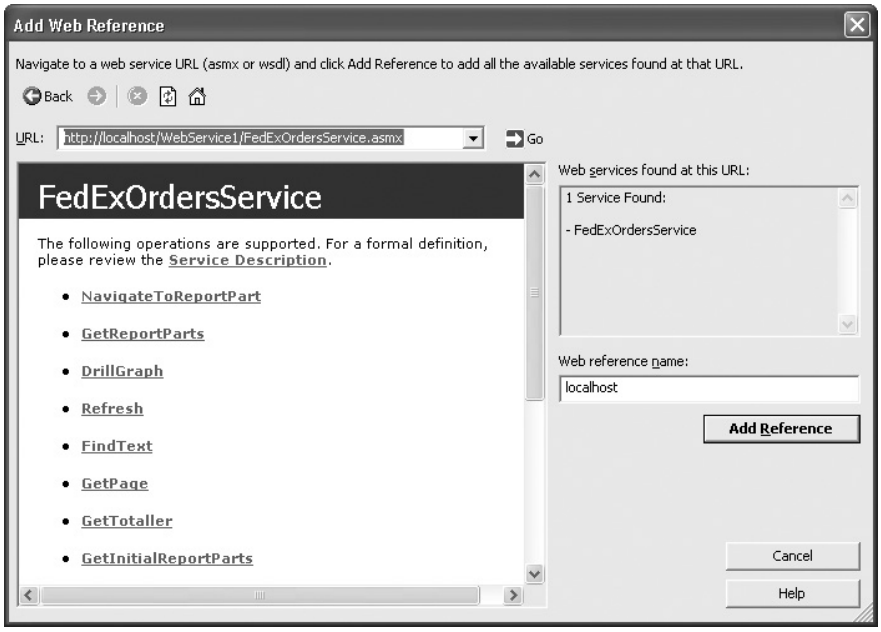
Binding by Adding the Report Web Service as a Reference

You may also add a reference to the web report service to your Windows project and instantiate a copy of any reports exposed by it. To do this, perform the following steps:

1. Add a reference to the web service by clicking the References category of the Solution Explorer. Right-click the References category and choose Add Web Reference from the pop-up menu. The Add Web Reference dialog box will appear.
2. In the Address line of the Add Web Reference dialog box, type the full URL for the web service. For example, to add a reference to the FedExOrders report web service discussed previously, you'd type:

   ```
   http://localhost/WebService1/FedExOrdersService.asmx
   ```

3. You'll see the same web browser display you would see if you navigated to the URL in a standard web browser. Also, a list of all exposed report web services will appear in the box on the right of the dialog box. And, the web server name will appear as the default name for the reference. If you wish to reference the web server by a different name, type it in the Web Reference Name text box. When you're finished, click OK to add the report web services as references.



To reference the web service in the Windows Forms Viewer, you may execute code similar to the following in the Load event of the form containing the viewer:

```
CrystalReportViewer1.ReportSource = _
New localhost.FedExOrdersService
```

When you run the application, the report exposed by the web service will appear in the form containing the viewer.

Setting Viewer Properties with Consumed Web Services

The actual report exposed by the report web service isn't "strongly typed." For this reason, you don't have the same control you do over a strongly typed report making use of the Report Engine namespace. However, you can still manipulate the report with classes exposed by the Crystal Windows Forms Viewer (you have a much richer set of choices if you've installed Crystal Reports 10 Developer or Advanced Developer editions).

For example, you can still make use of viewer properties to provide database credentials, set the record selection formula, or supply parameter field values. For example, the following code will supply a beginning and ending date to a date range parameter field included in the report exposed by the report web service:

```
Dim Parameters As New CrystalDecisions.Shared.ParameterFields
Dim Parameter As New CrystalDecisions.Shared.ParameterField
Dim ParamValue As New CrystalDecisions.Shared.ParameterRangeValue
```

```
Parameter.ParameterFieldName = "Date Range"
ParamValue.StartValue = #1/1/2002#
ParamValue.EndValue = #12/31/2002#
Parameter.CurrentValues.Add(ParamValue)

Parameters.Add(Parameter)

CrystalReportViewer1.ParameterFieldInfo = Parameters
```

Distributing Crystal Reports Windows Applications

Once you've designed your Crystal Reports Windows application, you'll probably need to package it for deployment to client PCs. Generally speaking, Crystal Reports–based applications are no different than any other application—you can create a typical Setup and Deployment project using standard VS.NET steps to create a Microsoft Installer package to install the Windows application on the target computer.

When deploying Windows applications that contain Crystal Reports, keep the following general concepts in mind:

- **The target machine must contain the .NET Framework** Depending on how you create your setup project, you may not include the .NET Framework files in the setup routines. Because Crystal Reports for VS.NET fully integrates with the .NET Framework, you'll need to ensure it is installed on the client computer before installing your Crystal web application.
- **You will need Business Objects–supplied merge modules** If you have installed Crystal Reports 10 Developer or Advanced Developer Edition on your development machine, you'll need to distribute updated Crystal Reports 10 modules with your application. This is accomplished by using one (and, if you are embedding geographic maps in your report, an additional) merge module when you build your setup project.

  Download the Crystal Reports 10 merge modules from http://support.BusinessObjects.com/MergeModules. You'll find included documentation that specifies what object models, viewers, and features are included with the various merge modules.

  If you are using the version of Crystal Reports that ships with Visual Studio .NET, merge modules are found in \Program Files\Common Files\Merge Modules. You may also find updated versions of these merge modules on the Business Objects web site.
- **Don't forget the keycode** When developing your reporting application, you may have noticed a Crystal screen prompting you to register your copy of Crystal Reports for VS.NET. This appears whether you are using the copy of Crystal Reports included with VS.NET or you've installed Crystal Reports 10 afterward (and, sometimes even if you've registered your stand-alone copy of Crystal Reports).

While you can bypass this screen when developing your application, you won't be so lucky when it comes time to distribute it. If a valid Crystal keycode isn't specified when you build your setup program, your distributed application won't work. You may obtain the

necessary keycode by following the steps in the registration prompt (you'll be taken to the Business Objects web site and asked to supply various pieces of personal information).

Once you're finished with registration, you can display the keycode by choosing Help | About Microsoft Development Environment from the VS.NET menus. Note the three-part keycode that appears next to the Crystal Reports entry in the list. Either write it down, or select the Crystal Reports entry and click the Copy Info button to copy the keycode to the Windows clipboard.

When you build your setup project, you *must* select the primary Crystal Reporting merge module and paste the keycode into the Keycode Properties box.

NOTE *If you installed Crystal Reports 10 Developer or Advanced Developer Edition, your keycode should be the same keycode on the program CD that you used when you installed stand-alone Crystal Reports 10.*