The ActiveX Control

The simplest Visual Basic (VB) programming interface to Crystal Reports is the Crystal ActiveX control. The ActiveX control is similar to Visual Basic extension (.VBX) controls found in earlier versions of Visual Basic. When added to your project, the Crystal ActiveX control will appear on the Visual Basic toolbox, along with other icons for command buttons, forms, radio buttons, and other Windows controls. By simply adding the control to a form in your VB project, you will be able to control report behavior using properties and methods, the same way you control other object behavior on the form.

Seagate has included the ActiveX control with Crystal Reports 8 for backward-compatibility with previous Crystal Reports versions. However, no new Version 8 functions or features have been added to the ActiveX control (actually, this control hasn't been updated since Crystal Reports 6). Seagate's primary focus for Visual Basic integration is the Report Designer component (RDC), covered in Chapter 23. If you're writing a new, from-scratch VB program and wish to integrate reports, take the little bit of extra time required to learn the RDC. As the sophistication of your VB program grows, you'll be able to take advantage of the extra features the RDC offers. However, this book still provides this information on the ActiveX control in case you're using an earlier version of Crystal Reports, or you really do want a quick, simple, and limited-function report integration solution.

Note: Seagate has added a section to Developer's Help that offers tips and directions for migrating an application from the ActiveX control to the RDC. Find Developer's Help in *Crystal Reports program directory*\Developers Files\Help. The filename is DEVLOPR.HLP. Look in the index for the topic "Migrating to the RDC from the OCX Index."

Adding the ActiveX Control

The first step to using the Crystal ActiveX control is adding it to your VB project. This process is similar to adding any other ActiveX control (also known as OLE Custom control) to your project:

1. Open an existing project or create a new VB project.
2. Choose Project | Components from the VB pull-down menus, or press CTRL-T. The Components dialog box will appear.
3. Scroll through the list on the Controls tab until you locate the Crystal Report Control item. Make sure you choose the Crystal Report Control, not other Crystal controls that may be registered on your system. If you don't find the Crystal Report Control, click the Browse button and locate and select the CRYSTL32.OCX file in the Windows system directory.
4. Click OK in the Components dialog box to add the Crystal ActiveX control to the VB toolbox, as shown in Figure 1.

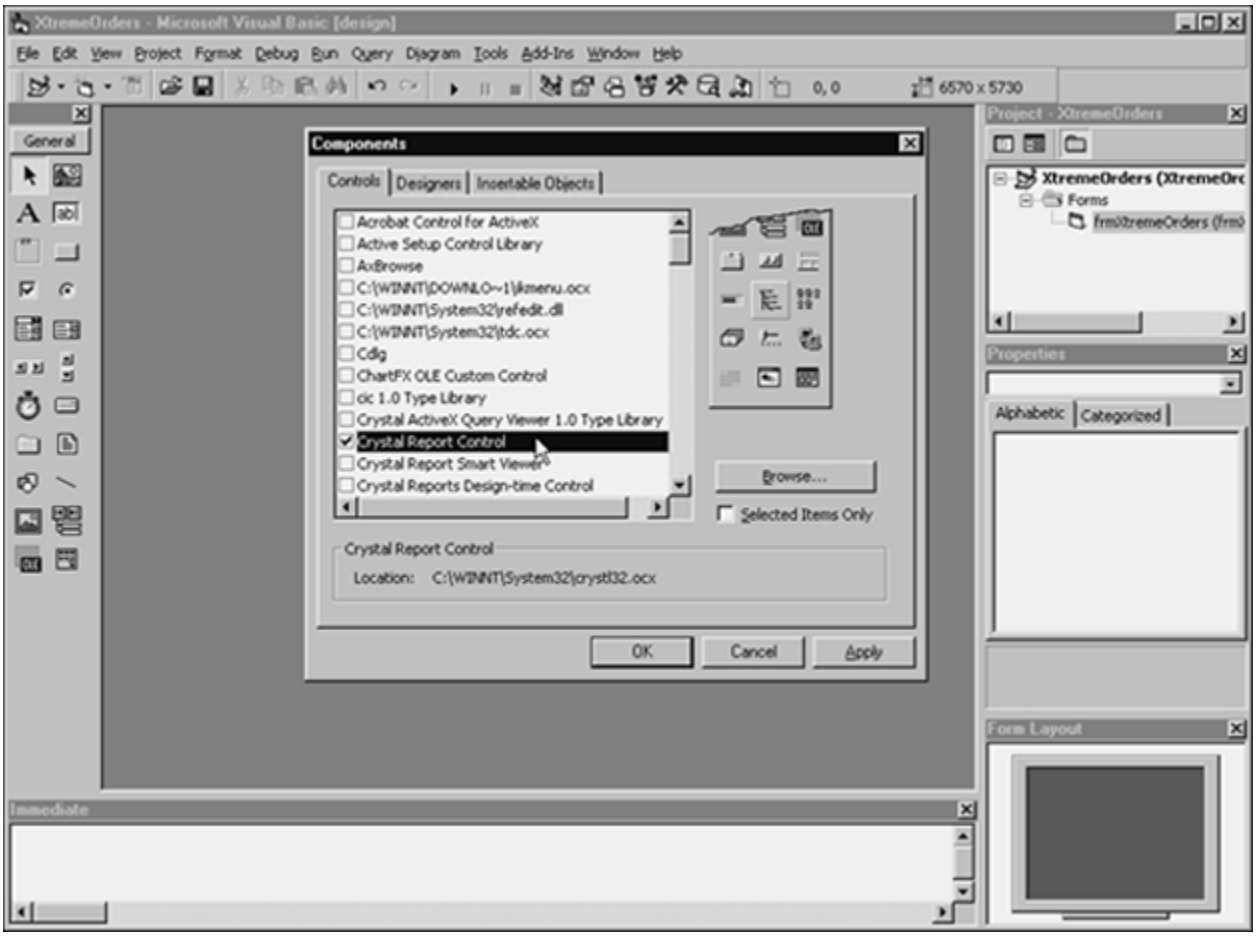


Figure 1: Adding the Crystal ActiveX control to the VB toolbox

Now you need to add the ActiveX control to a form in your project. Simply open the form you want to add the control to, and double-click the report icon in the toolbox. A dialog box will appear indicating that the ActiveX control is being provided by Seagate only for backward-compatibility (as discussed earlier in the chapter). Click OK. The icon will be added to the center of the form. You can drag the icon to an out-of-the-way place on the form if you wish. Remember that the icon is a design-time icon only—it won't appear on the form at application run time.

Typically, you'll want to add the control only once to one form. You can use the control from other forms and from module-level code by preceding the control with the form name, so you won't need to add the control to all forms in your project where you may want to display a report (see Figure 2).

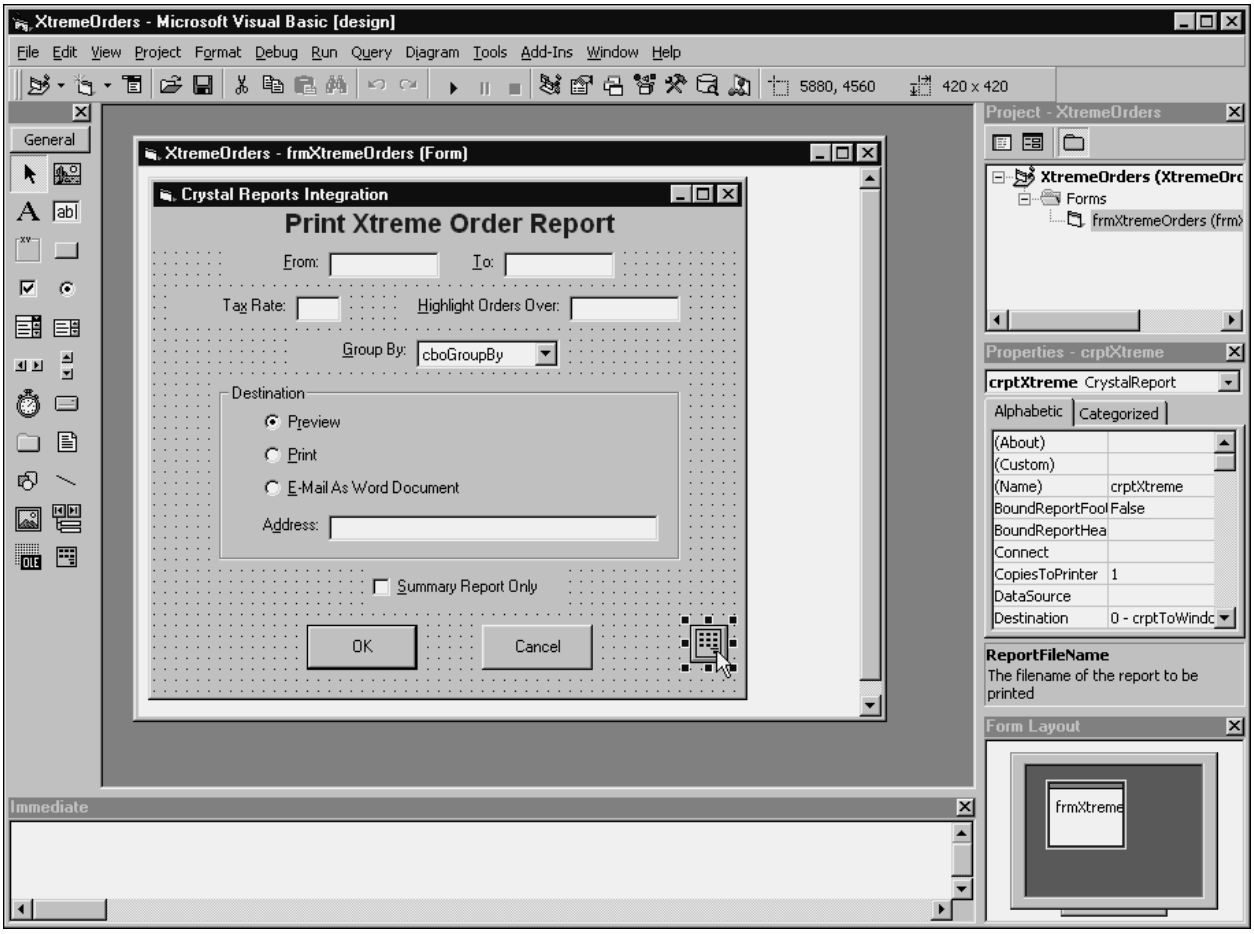
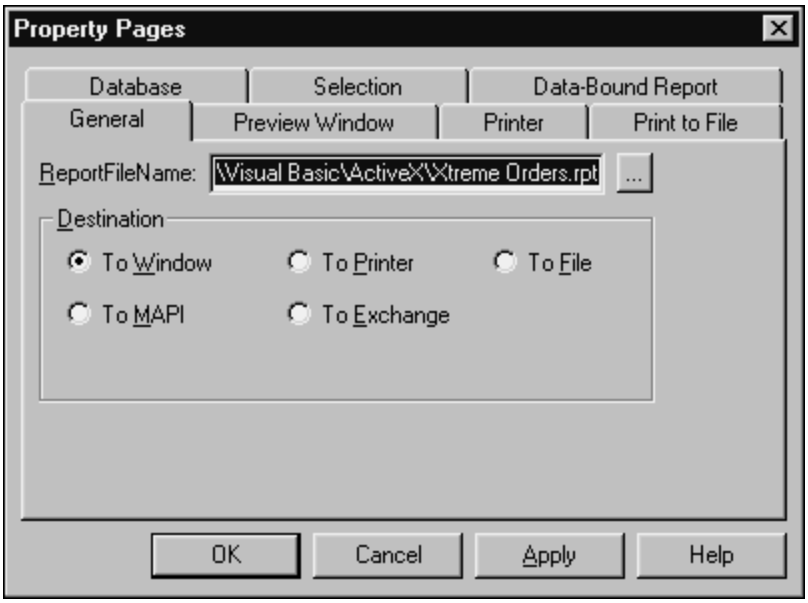


Figure 2: ActiveX control and properties

Once you've added the control to a form, select the control to display and change design-time properties in the VB Properties box. You'll notice that many features of Crystal Reports are available in the Properties box. You can modify report object properties in this Properties box the same way you do for other VB objects at design time.

In addition, the ActiveX control includes a Property Pages dialog box that organizes many ActiveX-control design-time properties in a simplified tabbed dialog box. To display it, select the (Custom) property in the Properties box and click the ellipsis (...) button next to it. The Property Pages dialog box will appear.



You'll notice many properties on the Property Pages dialog box that duplicate those in the VB Properties box. You can set these values in either place—it's entirely up to you.

The initial setting you may want to make with the new report object you've just added is the object name. Use the (Name) property in the Properties box to make this choice. Although the default name of CrystalReport1 will probably suffice, you might want to give the object a more meaningful name, including a standard object prefix for a Crystal Report object. You'll notice the object is named crptXtreme in the sample application.

At its most basic level, the ActiveX control can display a report in a preview window by just setting a few properties and adding one line of code to a VB form, menu option, or other control. Initially, you'll only need to specify the name of the Crystal Report .RPT file to use. This

name is set with the ReportFileName property in either the Properties box or the Property Pages dialog box.

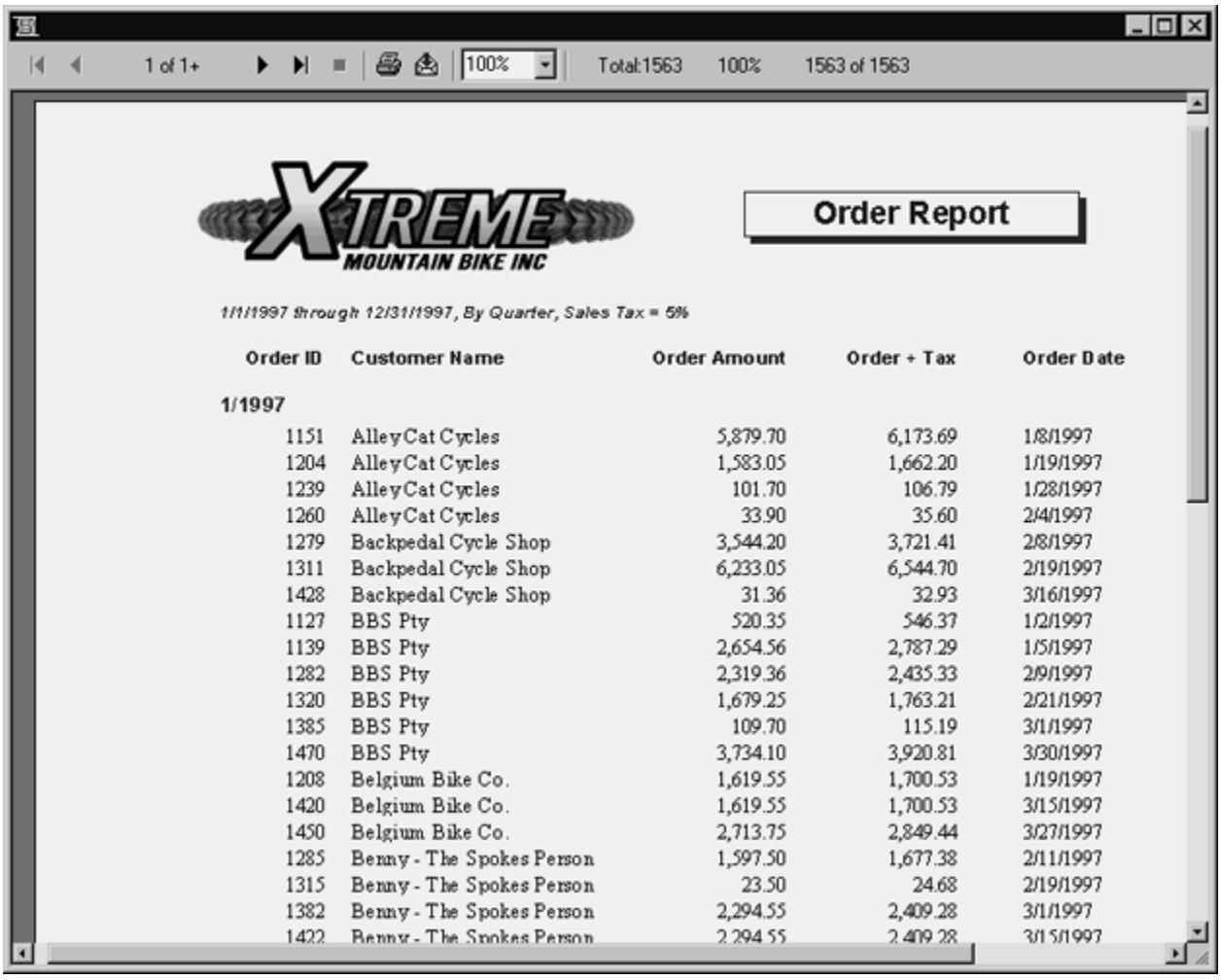
After you set the name of the report file, you can display the report merely by executing the control's PrintReport method:

```
intResult = crptXtreme.PrintReport
```

or by setting the control's Action property to 1:

```
crptXtreme.Action = 1
```

If the report can be found and run without any errors, it will be displayed in its own separate preview window.



The difference between the two approaches is the way errors are handled. By using the PrintReport method, a result code is returned indicating whether or not the report processed. A result code of 0 indicates that the report processed properly. If the method results in an error, the result code will return an error in the 20*XXX* range. However, setting the Action property to 1 doesn't return a result code. If an error occurs, a VB error in the 20*XXX* range will be "thrown," and it must be trapped by an On Error Goto routine. Crystal ActiveX control error handling is discussed in more detail later in the chapter.

Tip: All of the ActiveX control properties and methods discussed in this chapter are well documented in Crystal Reports Developer's Help. You may just want to open it in a separate Help window and minimize it for use while designing your Crystal Reports applications.

Customizing the Preview Window

When the preview window first appears, you may notice some default behavior that doesn't please you. For example, a group tree may not appear, even though you created groups on your report. And, if you specifically designed your report to facilitate drilling down by double-clicking group names or summary fields, you may be frustrated to discover that drilling down won't work.

This is simply the default behavior of the ActiveX control preview window, and it can be changed at design time or run time. Most properties that affect the behavior of the preview window begin with the word Window. For example, WindowTitle sets the text that appears in the title bar of the preview window. WindowShowGroupTree determines whether or not the group tree will appear. And, WindowAllowDrillDown determines whether or not you can drill down on the report. These properties can be set in the Properties box, in the Property Pages dialog box, or at run time. The following code sets these properties at run time:

```
'Customize preview window
crptXtreme.WindowShowGroupTree = True
crptXtreme.WindowTitle = "Xtreme Orders Report"
crptXtreme.WindowAllowDrillDown = chkSummary
```

This will customize the preview window to show the group tree, provide a title to the preview window, and allow drilling down based on the true/false state of a check box on the form.

There are many other properties that you can set to further customize preview window behavior, such as hiding preview window buttons and presetting the size and location of the preview window.

Passing Parameter-Field Values

Of the many features of Seagate Crystal Reports, parameter fields are one of the most flexible (see Chapter 12 for complete details). They allow a report viewer to select certain options when the report is run, such as choosing only certain records or affecting the way the report appears by using conditional formatting. Whatever value the viewer enters at the parameter-field prompt is passed into the report for use in formulas or record selection.

When you integrate a report containing parameter fields into a VB application, you have several choices as to how you want parameter fields to be handled by the application. By default, the ActiveX control will display the parameter-field prompts in a separate window and use the results in the preview window. In simple report applications, this may be sufficient.

However, one of the main reasons you'll probably be using Visual Basic to integrate the report is to provide much more control over how information is supplied to the report. You'll most likely want to gather information from the user with your own user interface and pass that information on to the report at run time. Since you can control both the report record selection and report formulas at run time from within your VB application, using parameter fields isn't as necessary with an integrated report as it might be for a stand-alone report running directly out of Crystal Reports.

However, if your report may be run in a stand-alone environment by some report viewers, as well as inside an integrated VB application by others, you still may need to pass values to parameter fields from within your code. Do this by setting the ParameterFields property at run time. The following is the syntax for setting this property:

```
[form.]Report.ParameterFields(ArrayIndex)[=ParameterName;NewValue;
SetCurrentValue]
```

This is the first of several property arrays exposed by the ActiveX control. A *property array* is used when more than one item's property can be set inside the report. In the case of parameter fields, a report can contain more than one item's property. By using a property array, you can set multiple parameter-field values in your code. This property array, as with all Crystal ActiveX control property arrays, is *zero-based.* That is, the first occurrence, or *element,* of the array is numbered 0, not 1. If your report contains four parameter fields that you want to change in the code, you'll set the ParameterFields property four times, using array indexes 0 through 3.

The property looks for a string value containing three arguments, separated by semicolons. The first, *ParameterName*, indicates the name of the parameter field you wish to change. The *NewValue* argument provides the actual value you wish to pass to the parameter field. The *SetCurrentValue* argument determines whether the parameter field will be prompted for a value, displaying as the default the value you pass.

Caution: Even if you pass parameter-field values, your report may not reflect the new values or may otherwise behave unpredictably if it contains saved data (that is, if Save Data with Report is checked on the Crystal Reports File menu). Since the overall purpose of a VB application is to run the report in real time after a user makes selections from the user interface, saving data with the report makes little sense anyway. If you are going to integrate a report with a Visual Basic application, turn this option off before saving your final version of the .RPT file.

The following code from the sample Xtreme Orders application uses the value of a text box to determine what is passed to the Highlight parameter field in the report. This parameter field is used in conditional formatting to add a light-blue background to orders that exceed the parameter-field value.

```
'Supply "Highlight" parameter field
If txtHighlight = "" Then
   crptXtreme.ParameterFields(0) = "Highlight;0;TRUE"
Else
   crptXtreme.ParameterFields(0) = "Highlight;" & txtHighlight & ";TRUE"
End If  'txtHighlight = ""
```

Here, the parameter-field value will be passed, and the usual Crystal Reports prompt will not be displayed.

In this case, the value is set to 0 if the text box is empty. This is important to note, because the actual parameter field in the report is numeric. Passing it an empty string will cause a run-time error when the report runs. Also, if you are passing a value to a date parameter field from your application, make sure it conforms to the Crystal Reports date syntax. You must pass **Date(yyyy,mm,dd)**, not mm/dd/yyyy, for the parameter field to work properly. Always pass a value to a parameter field in the exact same format that you would use when typing the value when prompted.

Caution: Remember—the ActiveX control doesn't expose Crystal Reports 8 advanced parameter-field features, such as multiple values or range values. You'll need to use another integration method to take full advantage of the Crystal Reports 8 parameter fields.

Controlling Record Selection

One of the most obvious benefits of integrating a Crystal Report into a VB program is controlling report record selection on the fly, based on your user's interactions with the VB application. Your application can present any type of user interface you desire, including controls populated by databases. When compared to Visual Basic controls, Crystal Reports parameter fields are very limited in their editing and validation capabilities. Text boxes and other VB controls can contain edit masks and other sophisticated validation features, such as the VB date picker, to help users choose correct options. You can then use the results of user actions to build a new Crystal Reports record-selection formula before the report is printed.

If you are used to using the Crystal Reports Select Expert for record selection, you need to familiarize yourself with the actual Crystal Reports formula that it creates, before you create a selection formula in your VB application. A Crystal Reports record-selection formula is a Boolean formula that narrows down the database records that will be included in the report. For example, the following record-selection formula will limit a report to orders placed in the first quarter of 1997 from customers in Texas:

```
{Orders.Order Date} In Date(1997,1,1) To Date(1997,3,31) And
{Customer.Region} = "TX"
```

Tip: For a complete discussion of Crystal Reports record selection and how to create Boolean formulas, consult Chapters 5 and 6.

There are two ActiveX control approaches to changing record selection: using the ReplaceSelectionFormula method, and using the SelectionFormula property. There is a *very* important distinction between the two. Using the ReplaceSelectionFormula method will completely replace any existing selection formula in the report with the one you pass, whereas setting the SelectionFormula property will *append* the selection formula you pass to any formula that already exists in the report. Be careful—only set the SelectionFormula property if you're sure you want to keep what's already saved with the report. Otherwise, the ReplaceSelectionFormula method will ensure your report uses only the selection formula you supply in your application.

The syntax for these options is as follows:

```
[form.]Report.SelectionFormula[= SelectionFormulaString$]

[form.]Report.ReplaceSelectionFormula [(SelectionFormulaString$)]
```

In each case, *SelectionFormulaString$* is either a string expression or a string variable containing the Crystal record-selection formula. Thus, to pass a date-range record-selection formula to the Xtreme Orders report, based on the contents of the From Date and To Date text boxes, use the following code:

```
'Supply record selection based on dates
strSelectionFormula = "{Orders.Order Date} in Date(" & _
   Format(txtFromDate, "yyyy,m,d") & ") to Date(" & _
   Format(txtToDate, "yyyy,m,d") & ")"
crptXtreme.ReplaceSelectionFormula (strSelectionFormula)
'Note: parentheses around strSelectionFormula are optional
```

Record-Selection Formula Tips

There are several important points to keep in mind when building a Crystal record-selection formula within your application. Specifically, there are some tricks to making sure that

your string values are formatted properly, and to making sure that as much of the SQL selection work as possible is done on the server rather than on the PC.

The string value you pass in your selection formula must adhere *exactly* to the Crystal Reports formula syntax. This includes using correct Crystal reserved words and punctuation. The previous example showed the necessity of building a Date(yyyy,mm,dd) string to pass to the selection formula.

It's also easy to forget the required quotation marks or apostrophes around literals that are used in comparisons. For example, you may want to pass the following selection formula:

```
{Customer.Region} = 'TX' And {Orders.Ship Via} = 'UPS'
```

If the characters TX and UPS are coming from controls, such as a text box and combo box, you might consider using the following VB code to place the selection formula in a string variable:

```
strSelectionFormula = "{Customer.Region} = " & txtRegion & _
" And {Orders.Ship Via} = " & cboShipper
```

At first glance, this appears to create a correctly formatted Crystal record-selection formula. However, if you submit this string with the ReplaceSelectionFormula method, the report will fail when it runs. Why? The best way to troubleshoot this issue is to look at the contents of strSelectionFormula in the VB Immediate window, by setting a breakpoint, or by using other VB debugging features. This will show the contents of the string variable after the preceding code has executed:

```
{Customer.Region} = TX And {Orders.Ship Via} = UPS
```

Notice that there are no quotation marks or apostrophes around the literals that are being compared in the record-selection formula, a basic requirement of the Crystal Reports formula language. The following VB code will create a syntactically correct selection formula:

```
strSelectionFormula = "{Customer.Region} = '" & txtRegion & _
"' And {Orders.Ship Via} = '" & cboShipper & "'"
```

The other major point to keep in mind is that if your report will be using a SQL database, remember that Crystal Reports will attempt to convert as much of your record-selection formula as possible to SQL when it runs the report. The same caveats apply to the record-selection

formula you pass from your VB application as apply to a record-selection formula you create directly in Crystal Reports. In particular, using built-in Crystal Reports formula functions, such as UpperCase or IsNull, and using OR operators instead of AND operators, will typically cause record selection to be moved to the client (the PC) instead of the database server. The result is very slow report performance. To avoid this situation, take the same care in creating record-selection formulas that you pass from your application as you would in Crystal Reports. Look for detailed discussions on record-selection performance in both Chapters 6 and 14.

You may also choose to create the SQL statement the report will use right in your VB application, and submit it to the report by setting the SQLQuery property. More information on this approach can be found by searching for **SQLQuery property** in Developer's Help.

Setting Formulas

Another powerful feature of Crystal Reports/Visual Basic integration is the ability to change report formulas from within your application at run time. As you might imagine, this opens up tremendous opportunities for a VB program to control the appearance and behavior of a report. This can be useful for changing formulas that are related to groups that may also be changed from within your code, formulas that show text on the report, or formulas that control math calculations or conditional formatting on the report.

Setting formulas at run time is similar to setting the record-selection formula at run time (described in the previous section). You'll need a good understanding of the Crystal Reports formula language to adequately modify formulas inside your VB code. If you need a refresher on Crystal Reports formulas, review Chapter 5.

The ActiveX control provides a property array that allows you to change one or more formulas in the report. The syntax for the Formulas property is as follows:

```
[form.]Report.Formulas(ArrayIndex)[= FormulaName= FormulaText]
```

As with parameter fields, formulas are available in a zero-based property array. You should begin using array element 0 for the first formula you are going to change, 1 for the second, and so on. You needn't change the Formulas property for all formulas in the report—just the ones you want to modify at run time. When you modify the Formulas property, you must

specify the exact formula name you wish to change, *without* the preceding @ sign, and then supply a string expression or variable that contains the new text you want the formula to contain.

In the Xtreme Orders sample application, two formulas are changed at run time, based on user-specified criteria on the Print Report form. First, the Order + Tax formula is modified, based on the user's entry in the Tax Rate text box. The formula is changed by the following code:

```
'Set @Order + Tax formula
If txtTaxRate = "" Then
   crptXtreme.Formulas(0) = "Order + Tax={Orders.Order Amount}"
Else
   crptXtreme.Formulas(0) = "Order + Tax={Orders.Order Amount} *"_
   & Str(txtTaxRate / 100 + 1)
End If  'txtTaxRate = ""
```

As in previous examples for parameter fields, you must take care when assigning a value to the formula. The Order + Tax formula is defined in Crystal Reports as a numeric formula. Therefore, you should pass it a formula that will evaluate to a number. If the user leaves the text box on the form empty, the program assumes there is no additional sales tax and simply places the Order Amount database field in the formula—the formula will show the same amount as the database field.

If, however, the user has specified a tax rate, the VB code manipulates the value by dividing it by 100, and then adding 1. This will create the proper type of number to multiply against the Order Amount field to correctly calculate tax. For example, if the user specifies a tax rate of 5 (for 5 percent tax), the VB code will change the value to 1.05 before combining it with the multiply operator and the Order Amount field in the formula. Notice also that the formula name *does not* contain the leading @ sign. This should be left off when supplying formula names to the Formulas property.

Another helpful use of formulas is to change text that appears on the report at run time, because the ActiveX control does not allow you to change the contents of text objects from within your code (this capability exists in the Report Designer component, covered in Chapter 23). Changing text that appears at run time is helpful if you want to display criteria that the user

has specified for the report, fields that the report is grouped or sorted on, or other useful information.

Consider the following code that modifies the Sub Heading formula on the Xtreme Order report. This formula, located in the page header, identifies the date range, grouping, and tax rates specified by the user.

```
'Set @Sub Heading formula
strSubHeading = "'" & txtFromDate & " through " & txtToDate
strSubHeading = strSubHeading & ", By " & cboGroupBy
If txtTaxRate = "" Then
   strSubHeading = strSubHeading & ", No Tax'"
Else
   strSubHeading = strSubHeading & ", Sales Tax = " & txtTaxRate & "%'"
End If  'txtTaxRate = ""
crptXtreme.Formulas(1) = "Sub Heading=" & strSubHeading
```

In this example, a string variable is used to build the correct formula syntax for the Crystal Reports string formula. Notice that the ultimate contents of the string variable must adhere to Crystal Reports syntax before the variable is passed to the report. In particular, this string formula begins and ends with an apostrophe. By using a string variable that is ultimately supplied to the Formulas property, you have more flexibility to examine the contents of the variable at a breakpoint in your code if you're unsure whether the correct syntax is being passed to the property.

Manipulating Report Groups

One of the requirements for the Xtreme Orders report is that the user be able to specify how the report is grouped. A combo box exists on the Print Report form that lets the user choose between Quarter and Customer grouping. In the Crystal Reports designer, this is normally accomplished by using the Change Group option to change the field a group is based on, as well as the order (ascending or descending) that you want the groups to appear in. If the group is based on a date field, such as Order Date, you can also specify the range of dates (week, month, quarter, and so on) that makes up the group. To familiarize yourself with various grouping options, refer to Chapter 3.

By being able to change these options from within an application at run time, you can provide great reporting flexibility to your end users. In many cases, you can create the appearance that several different reports are available based on user input. In fact, the user will be calling the same report, but the grouping will be changed at run time, based on user input.

Grouping is modified at run time by setting the GroupCondition property. This property allows you to change most of the group specification (with the exception of Specified Order grouping) that you can set in the Change Group dialog box in Crystal Reports. GroupCondition, as with other previously discussed properties, is a zero-based property array. And, as with other similar properties, you needn't specify a GroupCondition property for every group on your report—only those that you want to change from within your application.

The following is the syntax for the GroupCondition property:

```
[form.]Report.GroupCondition(ArrayIndex%)
[= group; field; condition; sortDirection]
```

The arguments for changing this property are as follows:

Group — A reserved word for each group contained on your report. The first group is referred to by the reserved word GROUP1, the second group by the reserved word GROUP2, and so on.

Field — The database field or report formula (including curly braces) that you want to base the group on.

Condition — A reserved word indicating the frequency of group creation. This is used for date field and Boolean field grouping to indicate how often a new group should be created. Allowed values for date fields are DAILY, WEEKLY, BIWEEKLY, SEMIMONTHLY, MONTHLY, QUARTERLY, SEMIANNUALLY, and ANNUALLY. Allowed values for Boolean fields are TOYES, TONO, EVERYYES, EVERYNO, NEXTISYES, and NEXTISNO. For nondate and non-Boolean fields, there is only one option you may supply for this argument, ANYCHANGE. Even though the ANYCHANGE argument doesn't truly affect grouping, you must still supply it.

Sortdirection The letters A or D. A specifies that groups will appear in ascending order, D specifies descending order.

All four arguments must always be supplied and must be separated by semicolons. The value may be supplied as either a string expression or a string variable.

Based on this syntax, the following code from the Xtreme Orders sample application will change the field the report group is based on, depending on the selection the user makes from the Group By combo box:

```
'Change Grouping
Select Case cboGroupBy
   Case "Quarter"
      crptXtreme.GroupCondition(0) = "GROUP1;_
      {Orders.Order Date};QUARTERLY;A"
   Case "Customer"
      crptXtreme.GroupCondition(0) = "GROUP1;_
      {Customer.Customer Name};ANYCHANGE;A"
End Select  'Case cboGroupBy
```

Changing Section Formatting

Because the design of Visual Basic applications often is intended to present reports to viewers in an online environment instead of just printing them on paper, interactive reporting features, such as drill-down, are invaluable to application designers. Having control over these capabilities at run time provides for great flexibility.

The Xtreme Orders sample application gives users the opportunity to specify whether or not they want to see the report as a summary report only. When this check box is selected, the VB application hides the report's details section so that only group subtotals appear on the report.

In addition, you'll need to control the appearance of the XTREME ORDERS.RPT file's two-page header sections (*Page Header a* and *Page Header b*), as well as two Group Header #1 sections (*Group Header #1a* and *Group Header #1b*). This is to accommodate two different sections of field titles that will appear differently if the report is presented as a summary report instead of a detail report. If the report is being displayed as a detail report, the field titles should

appear at the top of every page of the report, along with the subheading and smaller report title. If the report is displayed as a summary report, however, you will only want the field titles to appear in the group header section of a drill-down tab when the user double-clicks a group. Since no detail information will be visible in the main report window, field titles there won't be meaningful.

Finally, you'll want to show Group Header #1a, which contains the group name field, if the report is showing detail data. This will indicate what group the following set of orders applies to. However, if the report is showing only summary data, then showing both the group header and group footer will be repetitive—the group footer already contains the group name, so showing the group header, as well, looks odd.

Therefore, you need to control the appearance of four sections when a user chooses the summary report option. Table 22-1 outlines how you should conditionally set these sections at run time.

Section	**Detail Report**	**Summary Report**
Page Header b (field titles)	Shown	Suppressed (no drill-down)
Group Header #1a (group name field)	Shown	Hidden (drill-down okay)
Group Header #1b (field titles)	Suppressed (no drill-down)	Hidden (drill-down okay)
Details (detail order data)	Shown	Hidden (drill-down okay)

Table 1: Section Formatting for Different Report Types

Now you must look at the available ActiveX control properties to find one that allows control of section formatting at run time. The only property the ActiveX control offers is SectionFormat. As with many other properties, SectionFormat is a zero-based property array that can be set as many times as necessary to format multiple sections of the report. The syntax is as follows:

```
[form.]Report.SectionFormat(SectionArrayIndex%)[= sectionCode; visible;
newPageBefore; newPageAfter; keepTogether; suppressBlankSection;
resetPageNAfter; printAtBottomOfPage; underlaySection; backgroundColor]
```

The best place to look for a detailed breakdown of all the SectionFormat property's arguments is Crystal Reports Developer's Help. Search for **SectionFormat property**, and then **SectionFormat**.

The arguments roughly equate to the check box properties you see when using the Format Section option in the Crystal Reports design environment (refer to Chapter 8 for more details). In a nutshell, arguments are broken down into three types:

- **Section-name argument** A reserved word (again, look at Developer's Help for details) that indicates the exact section of the report you want to format
- **On-off (true-false) properties** Can all be supplied the letter *T* to turn the property on, the letter *F* to turn the property off, or the letter *X* to leave the property setting as it was when the report was created
- **Background-color property** Requires an RGB (red-green-blue) number consisting of three numerals from 0 to 255, inclusive, separated by periods (for example, 255.0.0 would set pure red as the background color for the section)

The following sample code changes a Crystal Reports ActiveX control named CrystalReport1. The code will show a second Group Header #2 section (Group Header #2b) that was suppressed when the report was created, set the New Page Before option on, and set the background color to pure blue.

```
CrystalReport1.SectionFormat(0) = GROUPHDR.1.1;T;T;X;X;X;X;X;X;0.0.255
```

Now that you've been given a fairly detailed overview of this property, how can it be used with the Xtreme Orders report to correctly format the report for summary viewing? If you refer to Table 22-1, you'll notice an immediate need to hide several sections, while still allowing drill-down functionality. In particular, the details section must be hidden, not suppressed. If the details section is suppressed, there's little sense in even creating a drill-down report in the first place.

Now, if you look at Developer's Help for available on-off arguments, you'll notice only the Visible argument. This equates to the Suppress (No Drill-Down) option from the Format Section dialog box in Crystal Reports. *No Hide (Drill-Down OK) argument is available with the ActiveX control!* You cannot format a visible section to be hidden, still allowing drill-down. As mentioned earlier, the ActiveX control is limited in its capabilities—you might not always be able to complete all the report customization you desire with this particular developer interface. So, then, how do you still accomplish your goal of providing the user a summary report versus a detail report choice?

A second .RPT file must be created to complete the ActiveX control integration example. This .RPT file will have the report sections preformatted as described in Table 22-1, so that the report will already display in a summary fashion. In addition, the regular .RPT file showing the detail report will be used. Based on the choice made in the Summary check box, the appropriate .RPT file will be assigned to the ActiveX control before any other properties are set. Here's the code from the sample application to accomplish this:

```
'Set report filename based on summary check box
'(because ActiveX control can't set Hide section property)
If chkSummary Then
   crptXtreme.ReportFileName = _
   "C:\Visual Basic\ActiveX\Xtreme Orders Summary.rpt"
Else
   crptXtreme.ReportFileName = _
   "C:\Visual Basic\ActiveX\Xtreme Orders.rpt"
End If  'chkSummary
```

Although this may not seem the most elegant method of accomplishing your ultimate goal, it is an acceptable method to deal with the inherent limitation of the Crystal ActiveX control. Subsequent chapters will illustrate that this limitation does not apply to other integration methods.

Choosing Output Destinations

Although viewing a report online in the preview window is a good way to interact with a report, you'll probably have situations in which you want the report printed to a printer, exported

to another file format, or attached to an e-mail message. These options are available from buttons in the preview window (if you haven't turned the options off with available ActiveX control properties). However, you may want tighter control over these capabilities from within your Visual Basic application. If you want to always send the output to a specific destination, you can set properties in either the Properties box or the Property Pages dialog box at design time to choose the output destination. Or, you may want to make this choice at run time based on user input.

In the Xtreme Orders sample application, radio buttons and a text box allow the user to choose whether to view the report in the preview window, print the report to a printer, or attach the report as a Word document to an e-mail message. If the user chooses e-mail as the output destination, an e-mail address can be typed into a text box. Once the viewer has made a selection, you need to set the output destination automatically in your VB code.

There are several properties that you can use to control the output destination: Destination, various EMail properties, and PrintFileType. The Destination property lets you make a general choice of output destination. The syntax is as follows:

```
[form.]Report.Destination[= Destination%]
```

Destination% is an integer number or predefined constant that determines the output destination. The integers and associated available constants are listed in Table 22-2.

Value	Constant	Description
0	crptToWindow	Sends the report to the preview window.
1	crptToPrinter	Sends the report to a printer.
2	crptToFile	Exports the report to a disk file. Specify the file format, such as Excel, Word, and so on, with the PrintFileType property, and specify the filename with the PrintFileName property.
3	crptToMapi	Attaches the report to an e-mail message using any MAPI-compliant e-mail client installed on the user's PC. Specify the file format for the attachment with the PrintFileType property.

6	crptToExchange	Exports the report to a Microsoft Exchange folder.

Table 2: Constants Used with the Destination Property

If you choose MAPI as the destination, the user's PC needs to have a MAPI-compliant e-mail client, such as Microsoft Outlook or Eudora Pro, installed. Also, you'll be able to set other options for the e-mail message, such as the To list, CC list, Subject, and message text using the ActiveX control's EMailToList, EMailCCList, EMailSubject, and EMailMessage properties. If you choose a destination of file or e-mail, choose the format of the exported or attached report (such as Excel, Word, Lotus 1-2-3, and so on) with the PrintFileType property. And, if you choose to export to a file, specify the filename with the PrintFileName property.

Tip: Additional properties are used to specify the field-separator character, as well as number and date formats used for reports exported to ASCII text files. Look at Developer's Help for more information.

In the sample Xtreme Orders application, the following code is used to choose an output destination, as well as to specify a file type and e-mail information, based on user selection:

```
'Set output destination
If optPreview Then crptXtreme.Destination = crptToWindow
If OptPrint Then crptXtreme.Destination = crptToPrinter
If OptEmail Then
   With crptXtreme
      .Destination = crptMapi
      .EMailToList = txtAddress
      .EMailSubject = "Here's the Xtreme Orders Report"
      .EMailMessage = "Attached is a Word document showing
the latest Xtreme Orders Report."
      .PrintFileType = crptWinWord
   End With 'cprtXtreme
End If  'optEmail
```

Displaying Print Options

By default, setting the destination to Printer and setting Action = 1 prints the report to the default printer without any further prompts. If, however, you'd like the user to be able to change

printers, choose the number of copies to print, or print only a certain range of report pages, you can display the Print Options dialog box with the PrinterSelect method. Examine the following code from the sample application:

```
'Let user change print options
If OptPrint Then crptXtreme.PrinterSelect
'WARNING: if a user cancels the Printer Select dialog box,
'the code has no way of knowing.
'The Action property will still be set!
```

As the remarks indicate, the one caveat to using the PrinterSelect method is that it doesn't return any result code. If the user happens to click the Cancel button on the dialog box, the VB code will continue merrily on its way, presumably to the Action property or PrintReport method, printing the report anyway.

If you prefer to design your own dialog box to gather print-related options from the user, you can use ActiveX control properties, such as PrinterDriver, PrinterName, and CopiesToPrint, to control printing. Look at Developer's Help for information on these, and other, print-related properties.

Error Handling

As with any Visual Basic program, you'll want to prepare for the possibility of errors that may occur. Integrating Crystal Reports with VB requires that you anticipate errors that the ActiveX control might encounter, in addition to other errors that the rest of your program may produce.

Recall from earlier in the chapter the two ways of actually processing the report once the ActiveX properties have all been set: the PrintReport method and the Action property. Also recall that the main difference between these two approaches is the way errors are handled. The PrintReport method returns a result code indicating 0 if the report ran correctly, or a code in the 20*XXX* range, containing an error code, if it didn't run correctly. Conversely, setting the Action property to 1 doesn't return a result code, but does result in a VB run-time error being thrown if the report doesn't print properly. In either case, you'll want to be prepared to intercept the potential error.

The choice of error method is really up to you. However, if you've already developed an On Error Goto routine to handle routine VB errors, it may be easier for you to simply add additional code to handle reporting errors. The error codes returned by either report-processing method will be in the 20*XXX* range. Using members of the Errors collection, such as Err.Number and Err.Description, you can handle errors or present meaningful error messages to the user. The *XXX* three-digit codes are specific to the Crystal Reports Print Engine, which is actually called by the ActiveX control. For a complete breakdown of these error codes, search Developer's Help for **Error codes, Crystal Report Engine**. There are a few additional error codes that apply only to the ActiveX control, and these can be found in Developer's Help by searching for **Error messages (ActiveX control)**. Examine the following code from the Xtreme Orders sample application. Notice that this code doesn't trap any particular reporting errors, except when the user cancels report printing or exporting (which throws a 20545 error). If another error occurs, this routine simply displays the error code and text in a message box.

```
Private Sub cmdOK_Click()
   On Error GoTo cmdOK_Click_Error

. . .

cmdOK_Click_Error:
   If Err.Number = 20545 Then
      MsgBox "Report cancelled", vbOKOnly + vbInformation, _
             "Print Xtreme Orders Report"
      Exit Sub
   End If   'Err.Number = 20545
   MsgBox "Error " + CStr(Err.Number) + " - " + Err.Description, _
          vbOKOnly + vbCritical, "Print Xtreme Orders Report"
End Sub
```

Caution: Often, you'll introduce errors before processing the report with the PrintReport method or the Action property, such as by submitting a syntactically incorrect formula or using an incorrect section name when formatting sections. However, these statements won't result in an error. Typically, no error will be detected until you actually process the report with PrintReport or Action = 1.

Other ActiveX Properties and Methods

This chapter has covered many of the typical Crystal ActiveX control properties and methods required for basic to intermediate report integration. The ActiveX control offers many additional features, however. To get an overview of other capabilities, search Developer's Help for **ActiveX control properties** or **ActiveX control methods**. In particular, you may want to explore additional features that are specific to SQL database handling or working with Crystal Reports subreport objects.

The Reset Method

The Reset method is handy if your VB program can be designed to print a report several times within the same code loop. In the Xtreme Orders sample application, for example, when the OK button is clicked, the dialog box is queried to set several properties for the ActiveX control, culminating in processing the report with Action = 1. However, when the report is finished printing or being e-mailed, or when the viewer closes the preview window, the Print dialog box remains displayed. The user can change options and click OK again.

But, what about previous property settings that were made from the last pass through the OK command button code? While your code may always set properties one way or another, regardless of the user interface, you may occasionally depend on the report's default behavior being in place.

For example, you may assume a certain default behavior for a report when it's initially loaded. If your code makes some change to this behavior, it will remain that way if you cycle back through your code a second time. However, you may want the report to automatically return to its default setting each time it runs, so you don't have to accommodate all the possible property changes that were made later in your code.

By using the Reset method, you can have most report properties return to their default .RPT file settings before properties are again set in code. This ensures that all properties are set to their defaults, in case your code cannot modify all properties that were previously changed. Here's the example from the Xtreme Orders application:

```
'Reset properties in case they were set in a previous instance
crptXtreme.Reset
```

SQL Database Control

Many corporate databases are kept on client/server SQL database systems, such as Microsoft SQL Server, Oracle, and Informix. Many Visual Basic applications provide front-end interfaces to these database systems, and they need to handle SQL reporting, as well. The Crystal ActiveX control contains several properties and methods that help when integrating reports based on SQL databases.

Logging On to SQL Databases

Because the VB application probably already handles SQL database security, and thus ensures that the application user has been validated by the database, you don't want the ActiveX control to require the user to log on to the database again when it comes time to print a report. By using the Connect or LogOnInfo properties, or the LogOnServer method, you can provide a valid database ID and password for a report from within the VB code. Search Developer's Help for information on these properties and methods.

Retrieving or Setting the SQL Query

When you submit a record-selection formula, as discussed earlier in the chapter, the Crystal ActiveX control will generate a SQL statement to submit to the database server automatically. However, if the record-selection formula contains Crystal formulas, an OR operator, or other characteristics that prevent Crystal Reports from including a WHERE clause in the SQL statement, report performance can be dramatically affected for the worse. You may, therefore, want to create the SQL statement the report uses directly in your VB application. As part of this process, you may find it helpful to retrieve the SQL statement that Crystal Reports is generating automatically.

To retrieve the contents of the existing SQL statement, use the RetrieveSQLQuery method. This populates the control's SQLQuery property, which can be examined inside your code. The SQLQuery property is a read/write property, so you can modify this property to make

changes to the query that the report will submit to the server. Search Developer's Help for specific syntax requirements for this method and property.

Note: As with Crystal Reports, you cannot modify the SELECT clause in the SQL query. Only FROM, WHERE, and ORDER BY clauses can be modified. Don't forget that you must still include the SELECT clause, however, and that it must not be changed from the original clause Crystal Reports created. Also, if you create an ORDER BY clause, you must separate it from the end of the WHERE clause with a carriage return/line feed (CR/LF) character sequence. Use the vbCrLf VB constant to add the CR/LF sequence inside the SQL query.

Reading or Setting Stored Procedure Parameters

If your report is based on a parameterized SQL stored procedure, you will probably want to supply parameter values to the stored procedure from within your code, much as you will want to respond to any Crystal Reports parameter fields that appear in other reports.

To retrieve existing stored procedure parameters, use the RetrieveStoredProcParams method, as illustrated here:

```
NumberofParams% = CrystalReport1.RetrieveStoredProcParams
```

Notice that this method returns an integer result indicating the number of parameters that the report contains. This method also populates the zero-based StoredProcParam property array with the contents of the parameters. You can then read the existing values if necessary. When you need to populate the parameters before running the report, set the StoredProcParam property with the new value, as illustrated here:

```
[form.]Report.StoredProcParam(ParameterArrayIndex%)[= newParameter$]
```

The *newParameter$* argument is a string expression or variable, regardless of the actual data type of the parameter. Just make sure to provide the proper type of data, such as only numeric information for a numeric parameter. Crystal Reports will automatically convert the string to the proper data type when it submits the parameter to the database server.

Subreport Control

If you've added subreports to your main Crystal Report, you may want a degree of control over how they behave from within your VB application, as well. The ActiveX control uses an interesting approach to let you customize many aspects of subreports in the same way you manipulate the main report.

If you already know the name of the subreport you want to change, simply modify the SubreportToChange property:

```
[form.]Report.SubreportToChange(= SubreportName$)
```

The *SubreportName$* argument is a string expression or variable that contains the name of the subreport you want to manipulate. Be careful—this argument is case-sensitive. After you set the subreport name, many properties that you set thereafter will apply to the subreport instead of to the main report. A list of the properties that will now apply to the subreport can be found by searching Developer's Help for **SubreportToChange property (ActiveX control)**. To again manipulate the main report, simply change the property again, supplying an empty string (" ") for the subreport name.

If your code needs to query the main report for the number and names of subreports it contains, use the GetNSubreports and GetNthSubreportName methods to make this determination:

```
[form.]Report.GetNSubreports
```

The preceding statement returns a zero-indexed number indicating the number of subreports in the main report.

```
[form.]Report.GetNthSubreportName (SubreportNum%)
```

This statement returns a string containing the name of the subreport that matches the SubreportNum index (zero-based). Make sure you pass an index between zero and what was returned with GetNSubreports.

Developer's Help contains complete information on these methods.

Tip: Complete information on using subreports in Crystal Reports is available in Chapter 11.