PROGRESSIVE

Whitepapers

Crystal Reports For VS.Net 2003 Startup Guide

June 2004

Version 1.0

Business Objects Starter Kit: Crystal Reports
Paul Delcogliano, Progressive Systems Consulting, Inc., http://www.progsys.com

Crystal Reports For VS.Net 2003 Startup Guide

Paul Delcogliano, Progressive Systems Consulting, Inc.

June 2004

Summary: This whitepaper provides step-by-step instructions on the details of generating and displaying reports in various deployment scenarios and application environments using Crystal Reports for Visual Studio.NET 2003.

Overview

What is the Crystal Reports For VS.NET 2003 Startup Guide?

The Crystal Reports For VS.Net 2003 Startup Guide (hereafter known as “the guide”) is intended to get competent .NET developers, who are new to Crystal Reports for Visual Studio.NET 2003 or familiar with older versions, up to speed quickly, using the product in a variety of reporting scenarios. The guide presents a scenario-driven approach to designing and building reports with Crystal Reports for Visual Studio.NET 2003. The guide walks through five different reporting scenarios. The scenarios increase in difficulty, and are described below:

Scenario	Description
Basic	A Tabular report, designed using the Expert, utilizing a standalone report file deployed as part of a Windows application. In this scenario, data from a single table is retrieved using OLE DB.
Intermediate	A Master-Detail report with drill-down capabilities, designed manually, utilizing an embedded report file and a ReportDocument object. In this scenario, the report is deployed as part of a Web application. Data is retrieved from views involving multiple tables using OLE DB.
Visual	A Graphical report, designed using the Expert, utilizing an embedded report file and a ReportDocument object. This report is deployed as part of a Windows application. Data is retrieved from a parameterized stored procedure using OLE DB.
Sub-report	A complex report that performs multiple queries through the use of sub-reports, designed manually, utilizing standalone report files. This report is deployed as part of a Web application. Data is retrieved from stored procedures using ADO.NET typed DataSets.
Document	A report based off a single result set row, used to produce a contract-style document, designed manually, utilizing an embedded report file and a ReportDocument object. This report is deployed as part of a Web application. Data is retrieved from a parameterized stored procedure using an ADO.NET typed DataSet. This scenario demonstrates programmatically exporting the report to PDF format that is then streamed back to the browser.

The scenarios are designed to illustrate the many powerful features of Crystal Reports for Visual Studio 2003, including:

- Integration with Visual Studio.NET 2003 (all scenarios.)
- Deployment in different application environments. The Basic and Visual scenarios demonstrate reporting in a Windows environment. The remaining scenarios illustrate reporting in a Web environment.
- Ability to report from multiple data sources, including typed datasets, views, tables and stored procedures. Typed datasets are displayed in the Sub-report and Document scenarios. Techniques using views and stored procedures are shown in the Intermediate, Visual, Sub-report and Document scenarios.
- Manual and wizard based methods for building reports. The Basic and Visual scenarios generate reports the report Experts. The Intermediate, Sub-report and Document scenarios create reports manually.
- Support for different report mediums, including report files (all scenarios) and ReportDocument objects (Intermediate, Visual, and Document scenarios).
- The Intermediate and Visual scenarios illustrate techniques for grouping and drill-down reporting capabilities.
- The Document scenario describes how to export files to different formats.

Prerequisites

The guide provides the most benefit to developers familiar with Visual Basic.NET and the Visual Studio.NET 2003 environment. All of the samples in this guide retrieve data from the Northwind database available with SQL Server. SQL Server is commonly used by developers writing applications with Visual Studio.NET. The techniques demonstrated to generate reports with SQL Server data are portable to other databases, although minor modifications may be required.

Sample Code

This guide discusses developing reporting solutions with Crystal Reports for Visual Studio.NET in depth. All of the sample code in the guide was developed using Microsoft ® Visual Studio.NET™ 2003 and was written in VB.NET.

The samples used in the guide build on one another, starting with the “Basic” scenario. In cases where details are given about a common process in a previous scenario, the current scenario will reference the previous discussion, rather than restating the process.

The samples accompanying this guide are divided into two projects, one for sample reports running in a Windows application, and one for sample Web applications. The Windows samples are located in the CRWindowsSamples project. The Web samples are located in the CRWebSamples project.

The sample code uses the Northwind database available with SQL Server 2000 or MSDE. For simplicity, a data link file is included with the sample code. All sample reports running in a Windows application use the data link file. All sample Web based reports connect to the database using “Database Expert” wizard provided by Crystal Reports. Creating a connection to the database is described in appendix A. Sample windows applications make database connections using a data link file. The data link file “Northwind.udl” is included with the sample code.

Scripts Required to Run the Sample Reports

The sample reports rely on custom stored procedures to run. In addition, all of the Web-based reports use a special SQL Server user account to connect to the database.

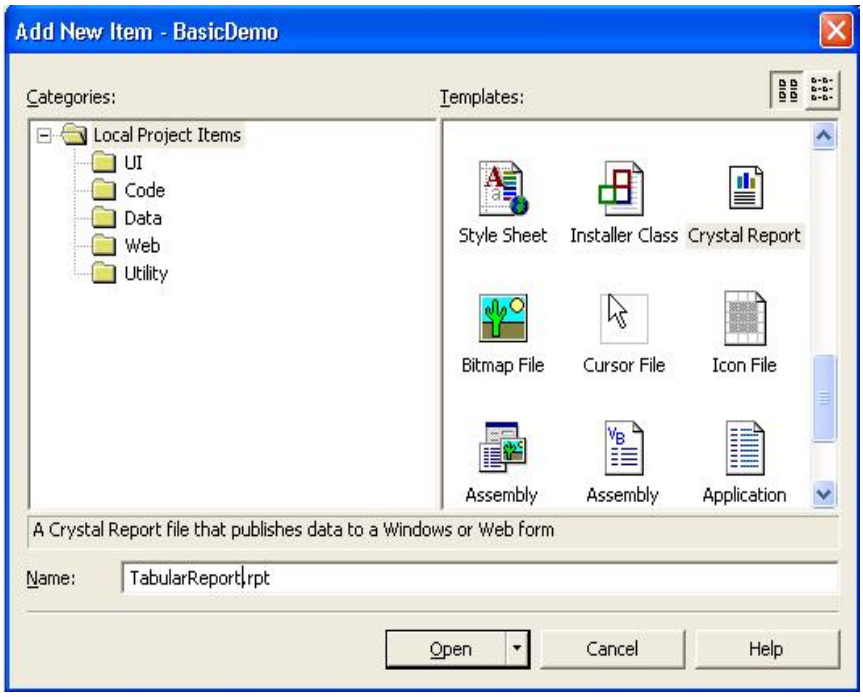
Prior to creating any of the sample reports, run the CreateSQLUserForSamples.sql script file to create the CRReports SQL Server user account and grant it access to the Northwind database. This script must be run by an account in the sysadmin fixed server role. Refer to the SQL Server Books Online documentation for more information about creating SQL Server accounts.

All of the stored procedures required to run the sample reports are found in the StoredProcsForSampleReports.sql file. As noted in the script file and the stored procedure comments, only an account with execute permission in the database will be able to create the stored procedure. Be sure to use a login account with sufficient privileges when running the script file.

Basic Reporting Scenario

The Basic Reporting scenario illustrates the simplicity of adding reporting capabilities to a Windows application. This scenario generates a report using the Crystal Reports design-time wizards, a.k.a. Experts, integrated with Visual Studio.NET. No code is necessary to generate this report. This scenario displays a report of Customers from the Northwind database in a Windows application.

To create the sample, create a new Visual Basic Windows Application Project in VS.NET. Name the new application “CRWindowsSamples”. Once the project is created, right-click on the project name in the Solution Explorer window and select “Add | Add New Item…” from the context menu. A window will appear, similar to the one in the figure below, presenting several object types that can be added to the project. Select the Crystal Report icon from the right side of the form and name the object “TabularReport.rpt”. Click the “Open” button to add the report to the project.



Meet the Experts

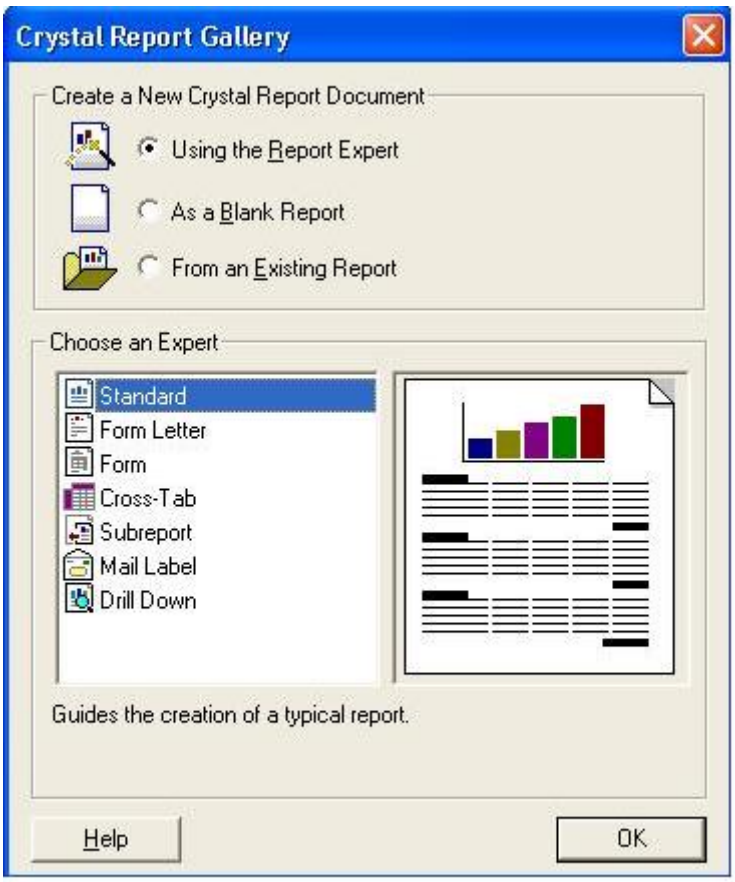
In addition to adding the report file to the project, clicking the “Open” button starts the “Crystal Report Gallery” wizard. This wizard presents three “Experts” for creating the report:

- Report Expert
- Blank Report
- Existing Report

The Report Expert guides report creation using a series of wizards and dialogs. There are several Report Experts to choose from including, Standard, Cross-tab, Sub-report, and Drill Down.

The Blank Report option creates a report from scratch.

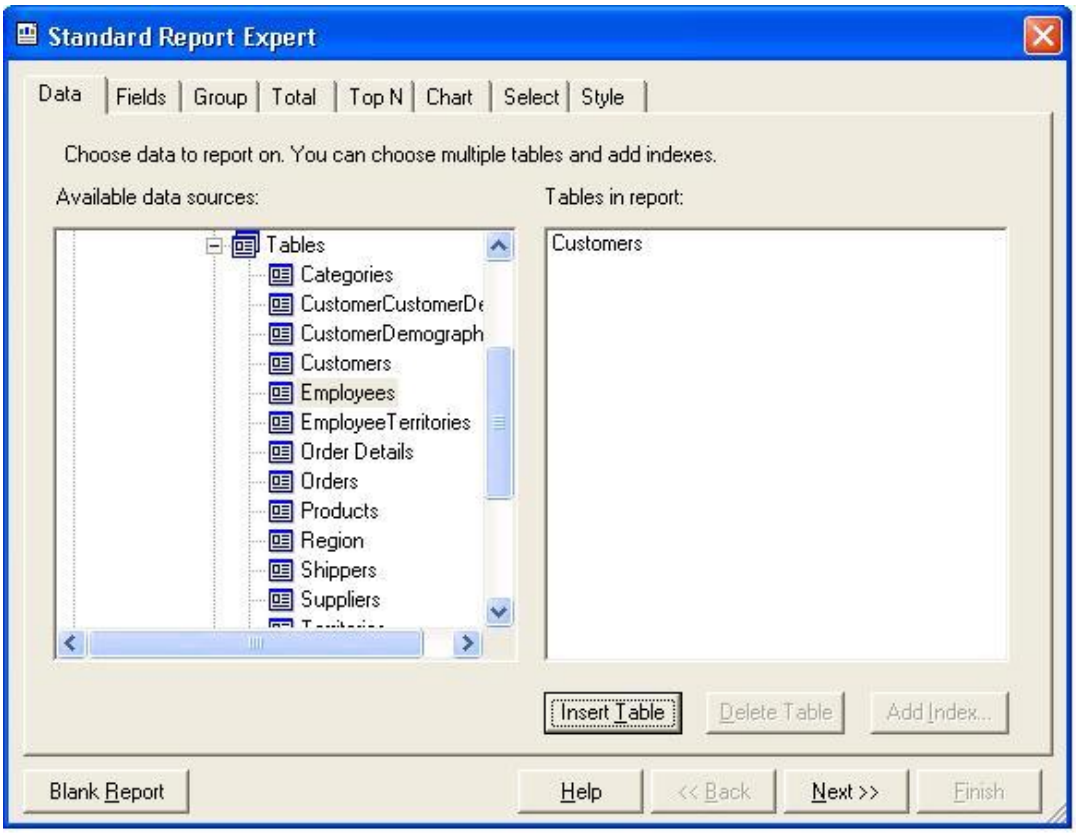
The Existing Report option creates a new report from an existing report. The figure below shows the Crystal Report Gallery wizard.



Choose the **Standard** report expert from the wizard, and then click **OK**. The Standard Expert creates a tabular report; data is laid out as columns and rows. Click **OK** to start the Standard Expert wizard.

As with most of the Experts, you must first make a connection to the data. There are a number of possible data sources including, OLE DB, ODBC, ADO.NET, Excel, etc. The section *Database Connection Used by Windows Based Reports* in appendix A describes how to establish the connection to the database using a data link file.

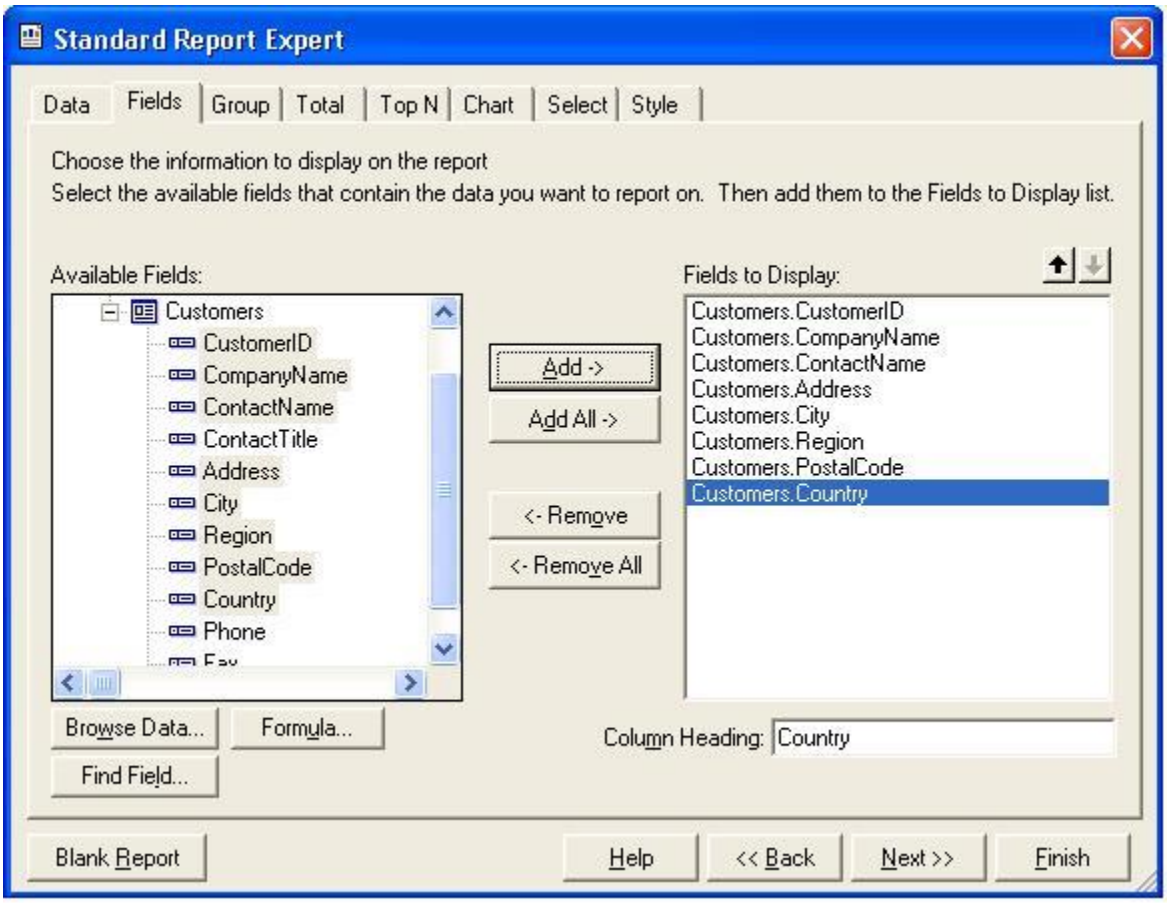
After the database connection is established, the Standard Expert wizard form will return showing the *Northwind* database node under the *OLE DB* node as an available data source. Expand the *Northwind* node to the *Tables* node. Select the *Customers* table from the treeview and click the **Insert Table** button. Clicking the button adds the Customers table to the report, as shown in the following screen shot.



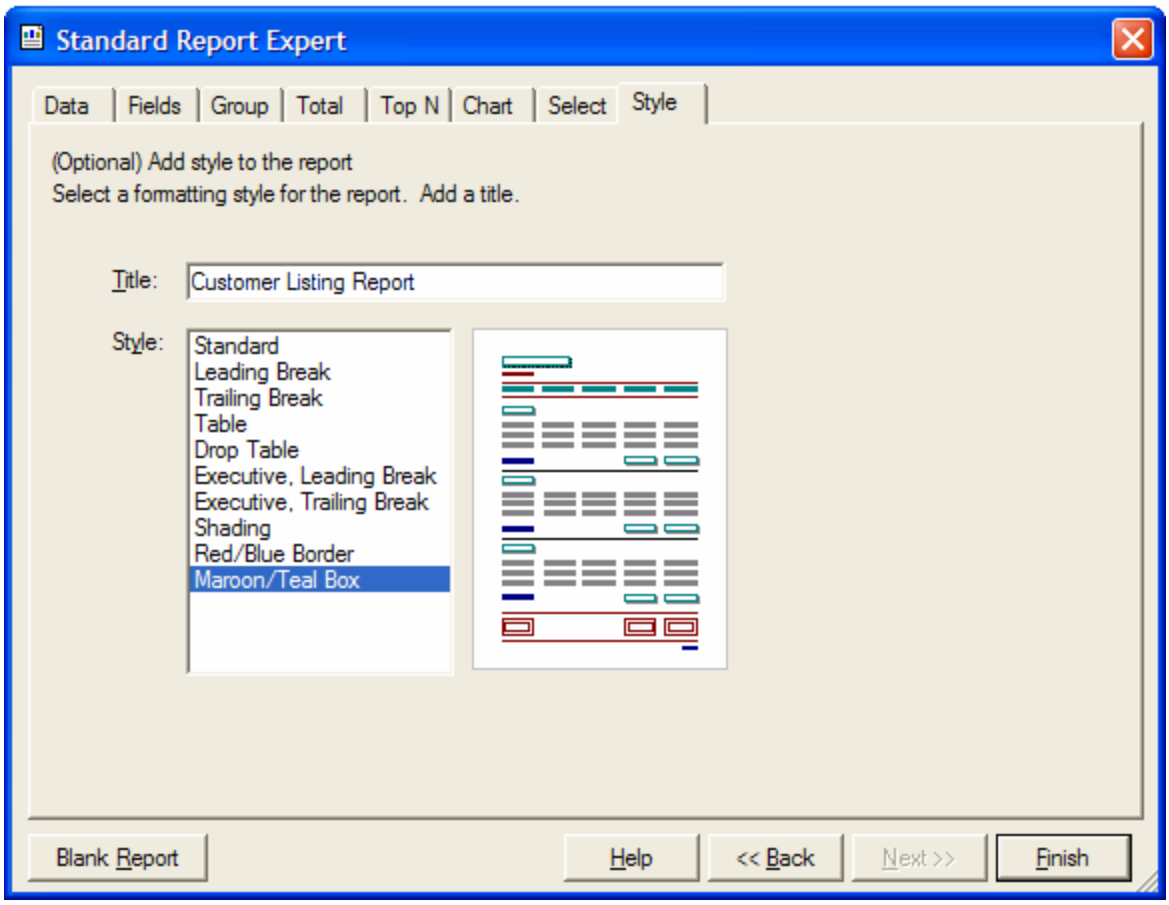
Specify which columns from the table should appear on the report. Click the **Next >>** button to select the columns to display on the report. Select the following columns:

- CustomerID
- CompanyName
- ContactName
- Address
- City
- Region
- PostalCode
- Country

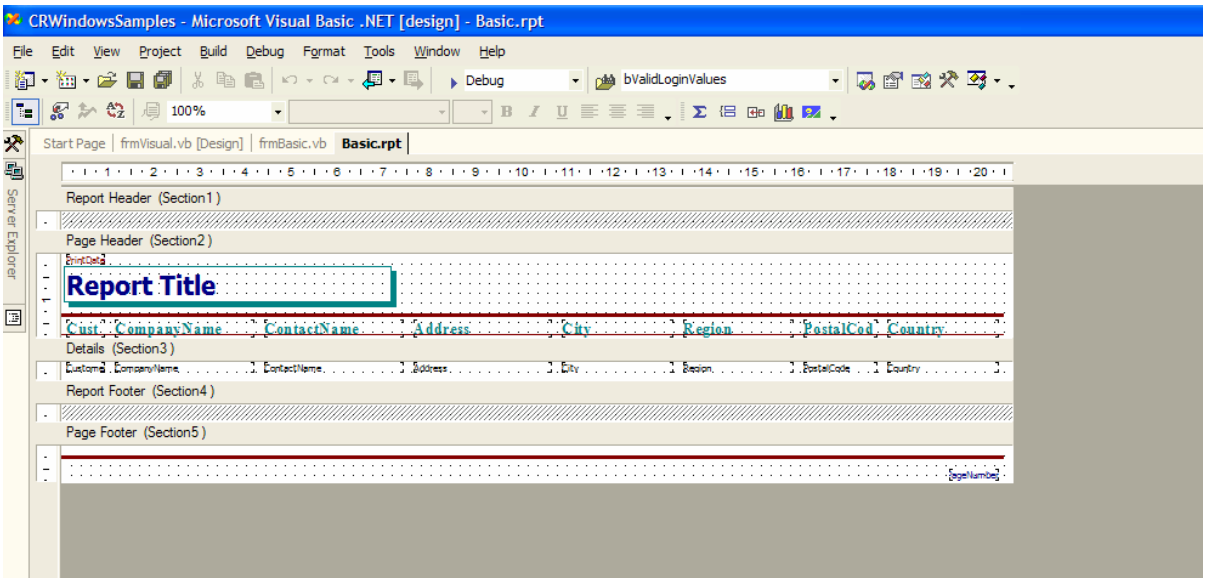
After selecting the columns, click the **Add ->** button. The Standard Expert form should look like the following:



Click on the **Style** tab to add a report title and to format the report's style. The selections on the Style tab are optional. The Expert provides a list of standard styles that can be applied to the report. Enter "Customer Listing Report" in the **Title** textbox. Choose the *Maroon/Teal Box* style from the list of styles available.



Click the **Finish** button to generate the report file. The report will be shown in design view within the Visual Studio.NET IDE. It should look similar to the following screen shot.



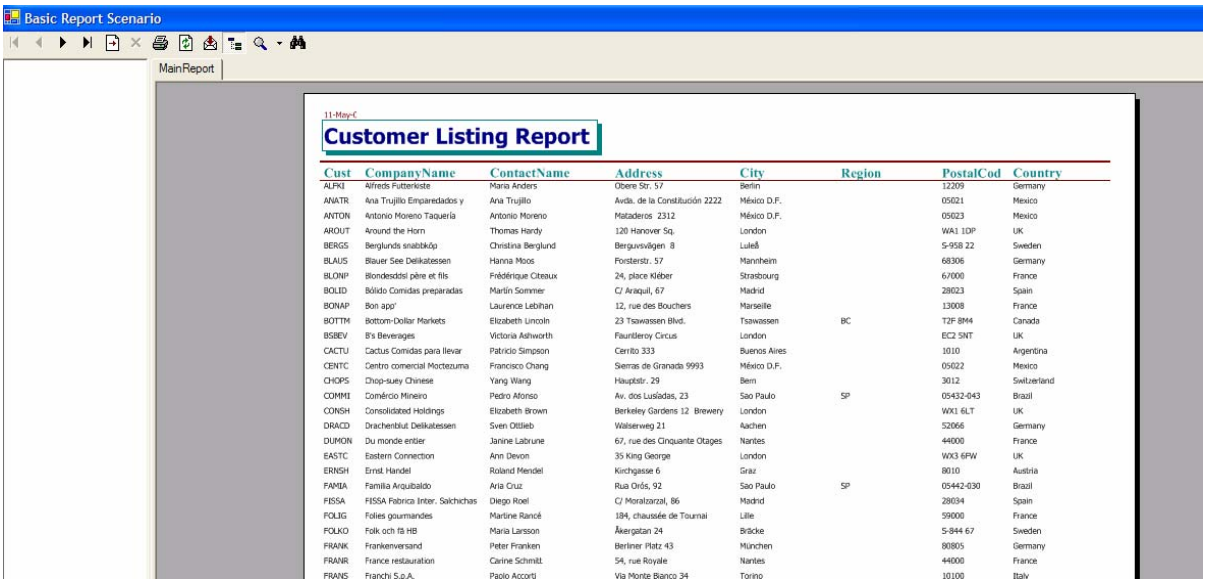
At this point, the report is created and ready to run. The last few steps involve setting up a form to host the report. Add a windows form to the project. Click on the tab named **Form1**. This will select Form1 so that the **CrystalReportViewer** control can be added to it. The CrystalReportViewer control displays a Crystal Reports report at runtime. Find the CrystalReportViewer control in the Toolbox (it is normally located under the *Windows*

Forms tab). Drag and drop the control from the toolbox onto Form1. Set the following properties of the CrystalReportViewer control:

Property Name	Property Value
Name	ReportViewer
Dock	Fill

There is one more property to set before this is a complete solution. The CrystalReportViewer's "ReportSource" property binds the report viewer control to the report file. Click on the **ReportSource** property in the Properties window. A drop down will appear allowing for the selection of the report file to bind to the viewer control. Browse to the **TabularReport.rpt** file and select it. The file should be located in the CRWindowsSamples's project folder.

The application is now ready to run. Press **F5** to start the application and display the report. The report output should look similar to the following:



The following list summarizes the steps taken in this section to create the sample report.

- Add a new Crystal Report object to the project.
- Layout the report using the Standard Expert
    - Connect to the database
    - Add a table to the report
    - Select the columns from the table to appear on the report
    - Set the report style
- Add a CrystalReportViewer control to the windows form in the project.
- Bind the report to the viewer by setting the viewer's ReportSource property to the report file.

Scenario Conclusion

The Basic scenario illustrates how easy it is to add reporting to a windows application. However, most real-world reporting requirements are more involved than simply listing values from a table. The “Intermediate” scenario walks through the creation of a report that is more commonly seen in application development, the master-detail report.

Intermediate Reporting Scenario

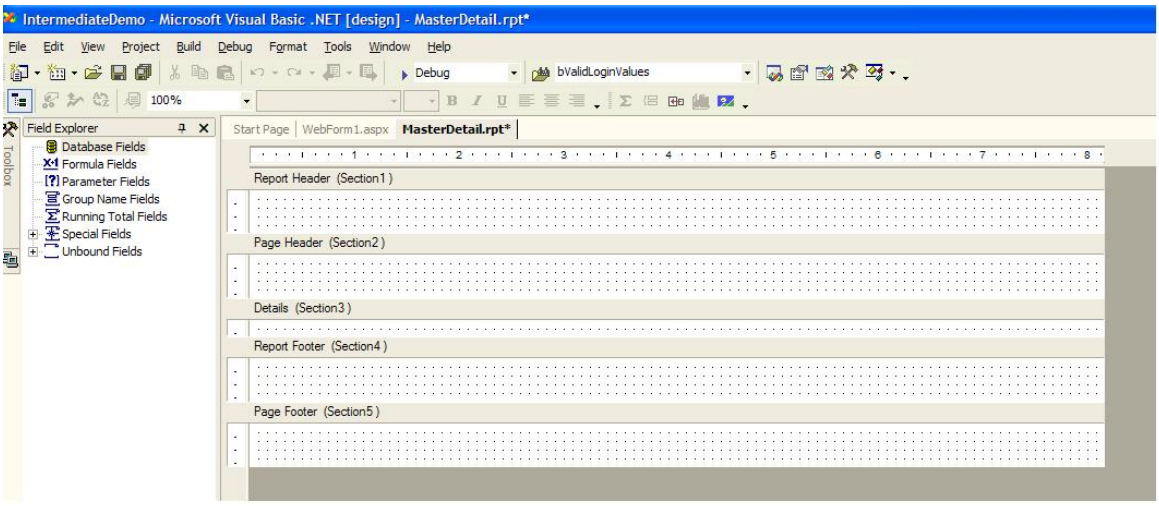
The Intermediate reporting scenario demonstrates how to manually create a web-based, master-detail report with drill-down capabilities. This scenario also shows how to bind the report to a ReportDocument object.

Delivering reports over the Web is a powerful feature. With Web based reporting, users have access to data 24 hours a day, 7 days a week, regardless of their location. Deploying reports over the Web is a straightforward and simple process, not much more complicated that the Basic Reporting scenario described above.

The report demonstrated in this section uses the “Products by Category” view from the Northwind database. The view returns category (master) and products (detail) data from the database. The report is grouped by category name, calculating the total number of products available under each category. Clicking on a category name expands the category and displays the list of products under the category. Each product is displayed with the number of units in stock.

To create the report, start by creating a new ASP.NET Web application. Name the new application “CRWebSamples”. Once the project is created, right-click on the project name in the Solution Explorer window and select **Add | Add New Item…** from the context menu. Select the Crystal Report icon from the right side of the dialog that appears, and name the object “MasterDetail.rpt”. Click the **Open** button to add the report to the Web project.

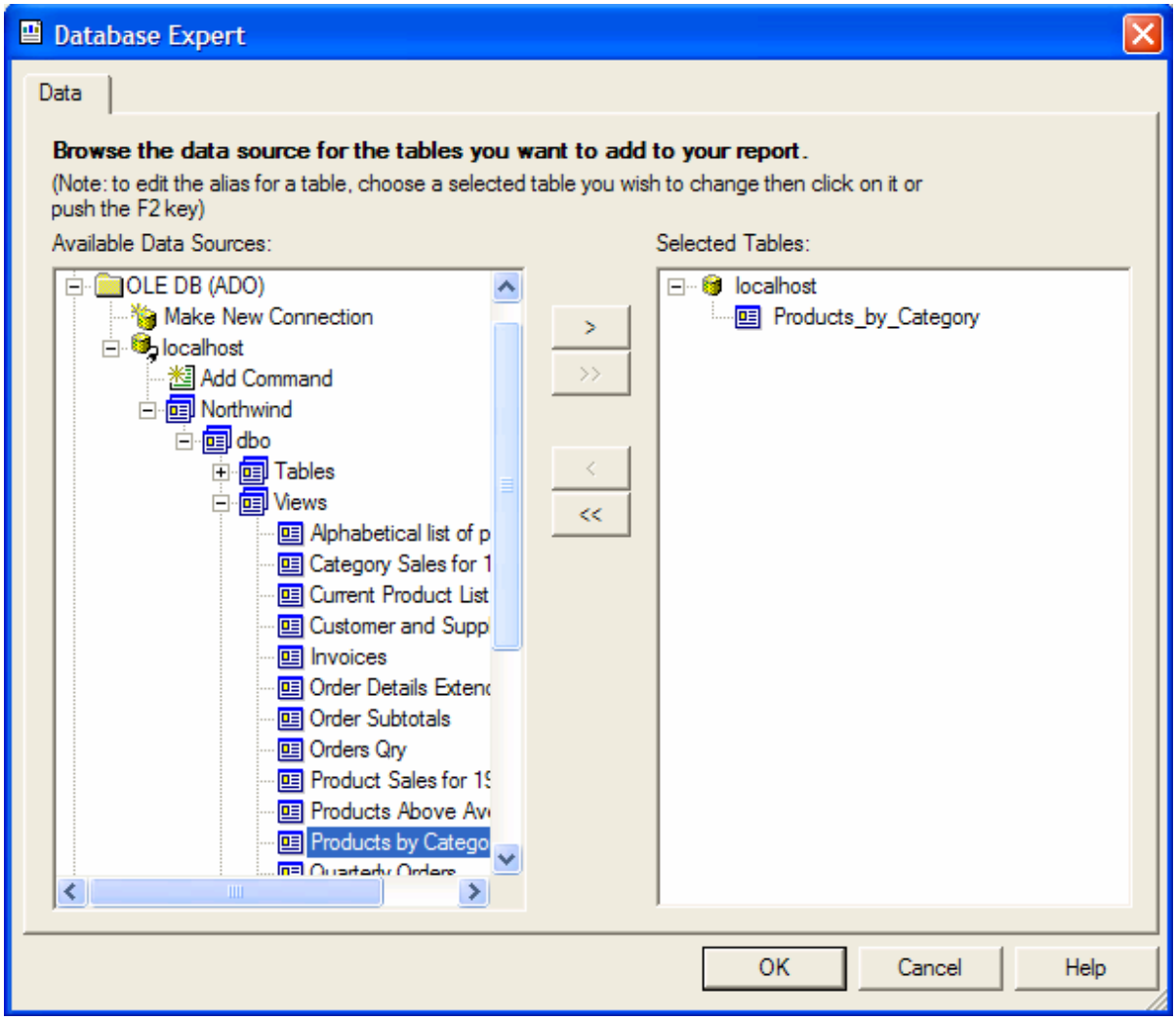
The report used for the Intermediate Reporting scenario is created manually using the Blank Expert. When the Crystal Report Gallery wizard loads, choose the Blank Expert option and click the **OK** button. A blank report form loads in the designer.



Right click on the report form in the designer to bring up a context menu. From the menu choose the **Database | Add/Remove Database…** option. Choosing this option starts the Database Expert wizard. The process of selecting a provider and connecting to the database is described in the section *Database Connection Used by Web Based Reports* in appendix A.

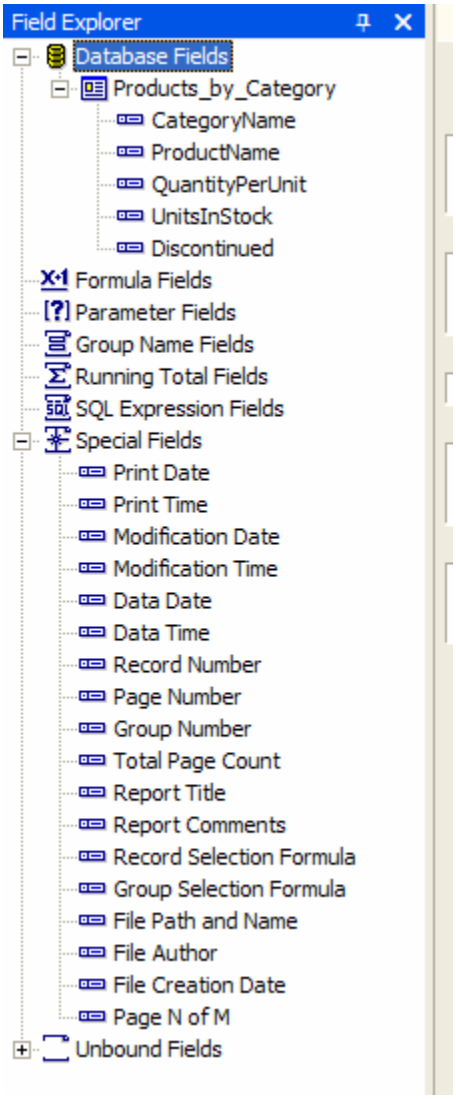
Now that a connection is established, expand the *Northwind* node from the available data sources list. Continue to expand the list down to the *Views* node. With the Views

node expanded, double click the *Products by Category* node. Now the view is listed under the *Selected Tables* list. Click **OK** to close the wizard.



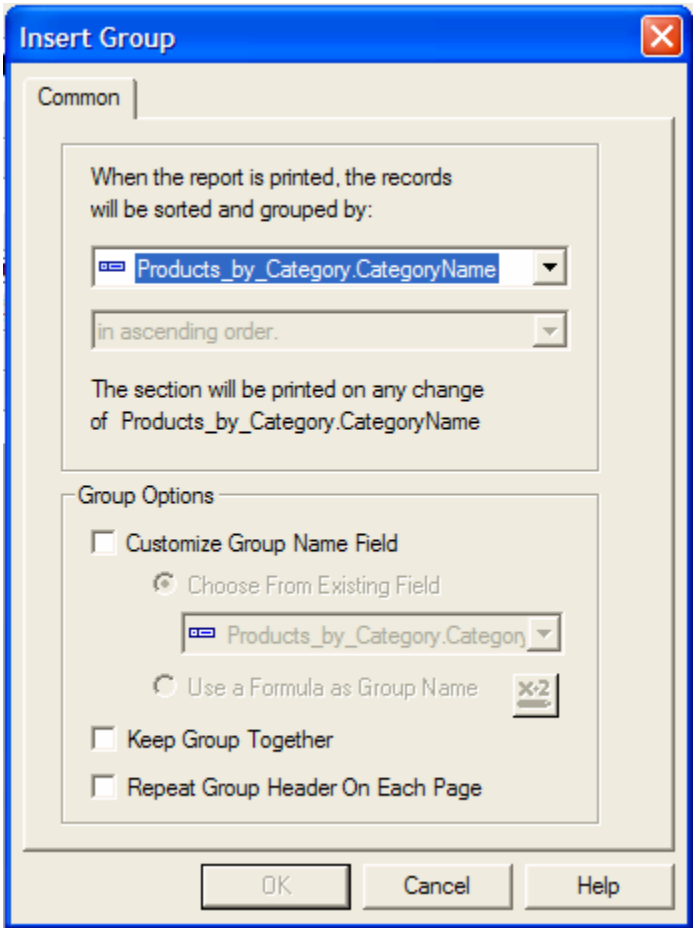
Build a Report from Scratch

All of the fields available from the *Products by Category* view are available from the *Field Explorer* window.



Building a report from scratch involves dragging fields from the Field Explorer onto the report designer. The following fields should be added to build the sample report.

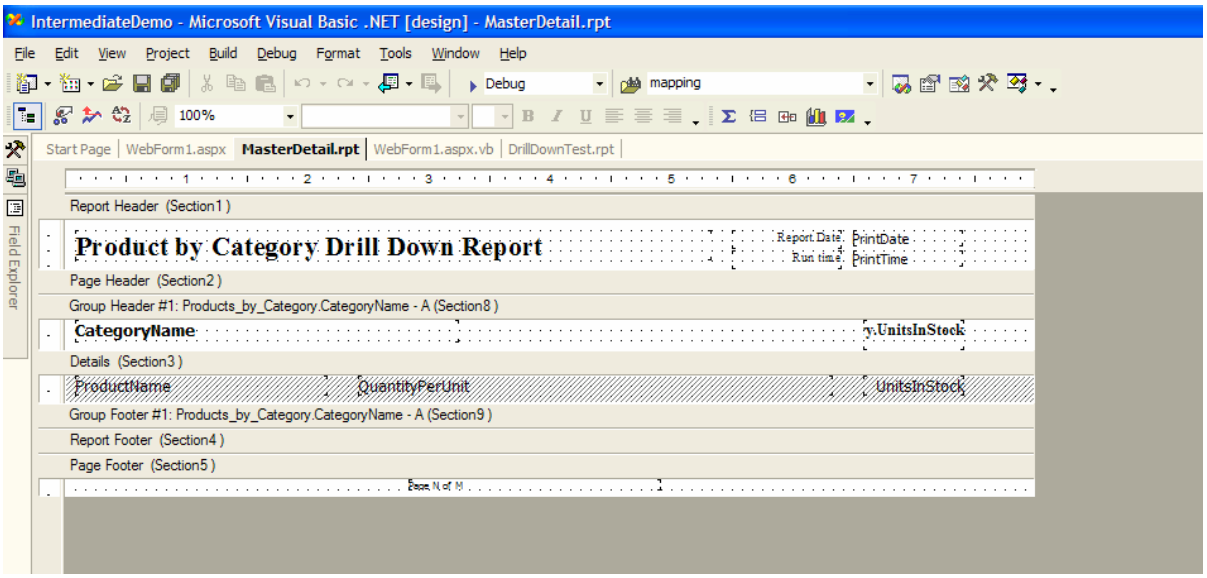
- Add the following fields to the Report Header Section
  - Text Object whose text is “Product by Category Drill Down Report”. Text object fields can be added to a report by right-clicking on the report and selecting **Insert | Text Object** from the menu.
  - Insert a *Print Date* field object by right clicking on the report and choosing the **Insert | Special Field > | Print Date** option or by dragging the Print Date field from the Field Explorer window.
  - Insert a *Print Time* field object by right clicking on the report and choosing the **Insert | Special Field > | Print Time** option or by dragging the Print Time field from the Field Explorer window.
- Add the following fields to the Group Header Section
  - Right-click on the report and choose the **Insert | Group…** option. The *Grouping* dialog will begin. Select the field that the report uses to group on, *Products_by_Category.CategoryName* from the drop down list. Click **OK** to add the group to the report.



- To display a total of the units in stock for each category, right-click on the report and choose the **Insert | Subtotal...** option. The *Subtotal* dialog will begin. Select the *Products_by_Category.UnitsInStock* from the *Insert a subtotal for the field* drop down list. Click **OK** to add the field to the report.

- Add the following fields to the report's Details Section
  - Individually drag the *ProductName, QuantityPerUnit,* and *UnitsInStock* fields from the Field Explorer and drop them onto the report's Details section.
  - Right click on the Details section. From the context menu, choose the **Hide (Drill-Down OK)** option. Setting this option configures the report to drill down into the Details section at runtime.

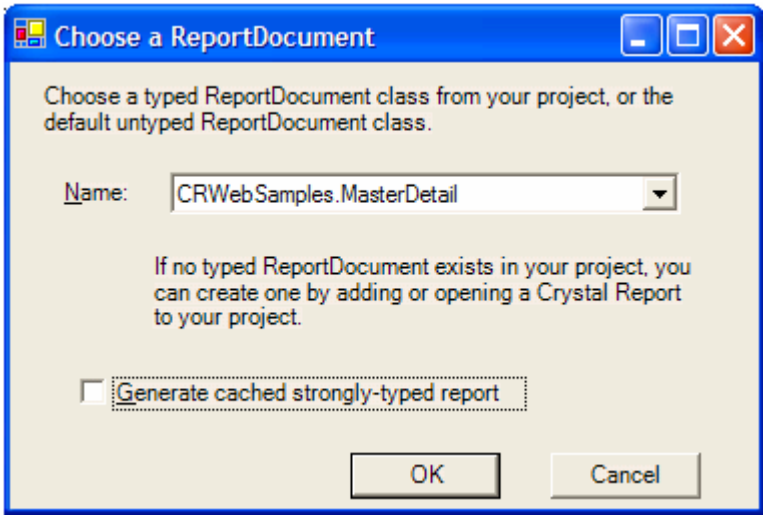
After all of the fields have been added to the report, lay them out similarly to the layout pictured below.



Once the report is laid out, it needs to be bound to the CrystalReportViewer control. Unlike the Basic report scenario, which bound the report to the control at design time, this scenario binds the report to a control at runtime. Click on the *WebForm1* tab in VS.NET. Rename WebForm1 to “frmIntermediate”. Drag and drop the CrystalReportViewer control from the toolbox onto frmIntermediate. Set its DisplayGroupTree property to “False”.

This scenario uses the ReportDocument control to bind the report to the CrystalReportViewer control. The ReportDocument class exposes properties and methods of a report at runtime. Access to report sections, options (report options, print options, etc.) and login credentials is available through the ReportDocument class.

The ReportDocument control is located on the *Components* tab of the Toolbox. Drag the control from the toolbox and drop it onto frmIntermediate. This will open a dialog, prompting for the typed ReportDocument class. Reports embedded in the project, like the MasterDetail report, are available from the drop down. Select the MasterDetail report. Uncheck the **Generate cached strongly-typed report** check box and click **OK**.

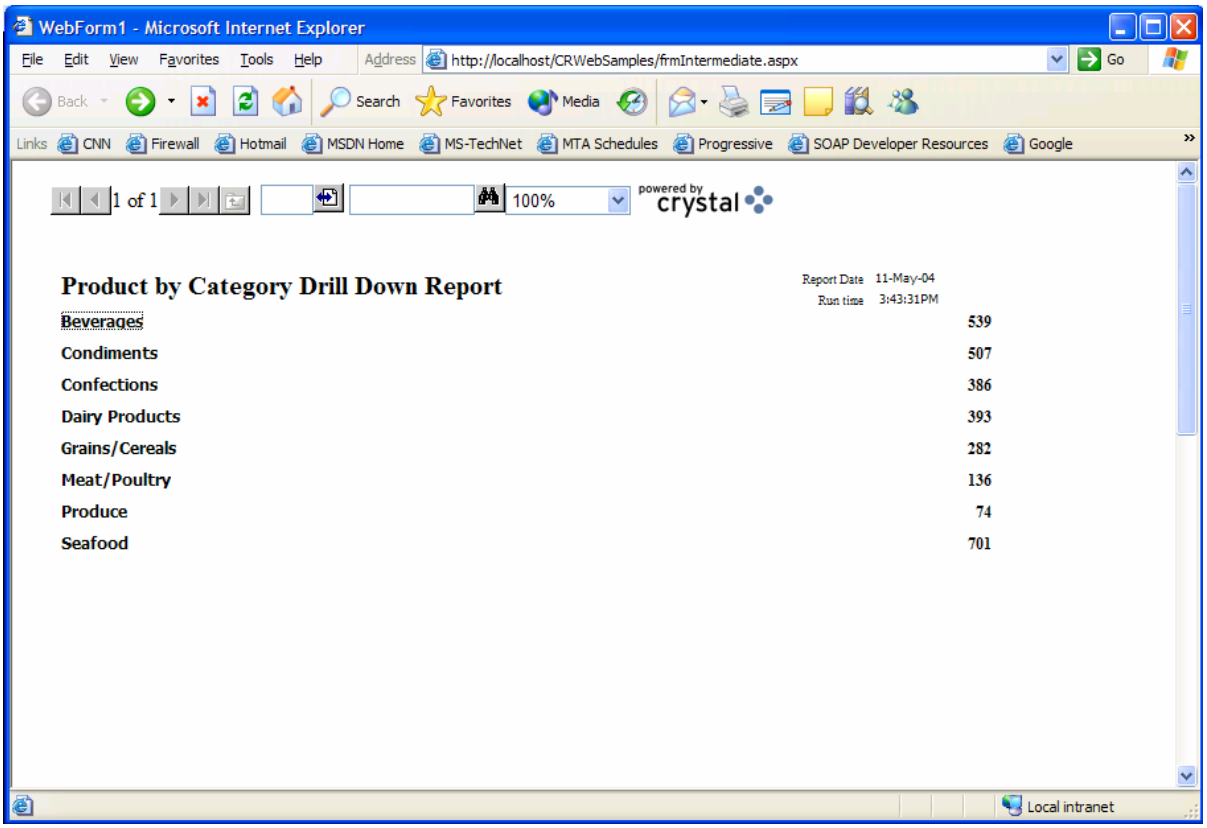


This creates a strongly typed ReportDocument named masterDetail1 and adds it to frmIntermediate. The final step is to bind the ReportDocument to the report viewer. Add the following lines of code to frmIntermediate's Page_Load event.

```
' set the database login information
masterDetail1.SetDatabaseLogon("CRReports", "crreports")

' bind the ReportDocument object to the report viewer. This will load
' the report in the "Intermediate" web page.
CrystalReportViewer1.ReportSource = masterDetail1
```

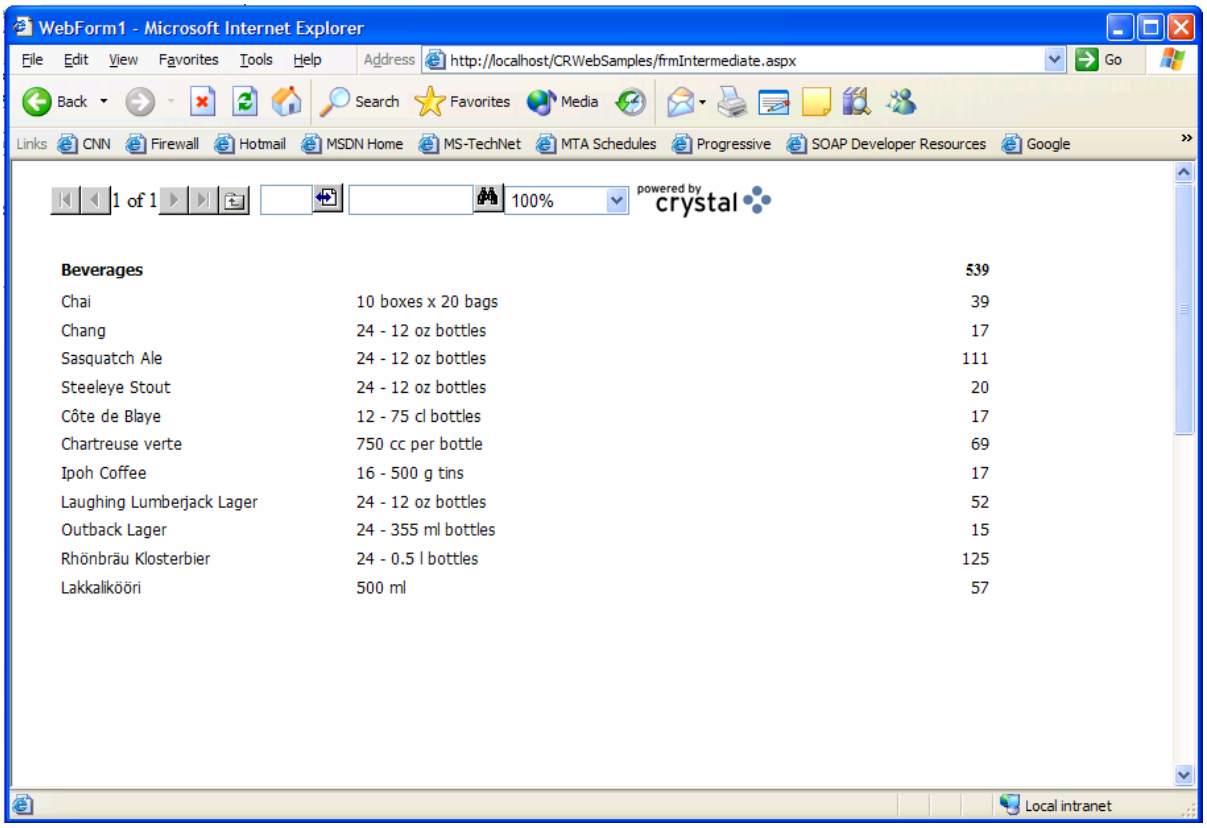
The ReportDocument's SetDatabaseLogin method passes login credentials to the views used for the report. If the login credentials are not set, or are invalid, an exception will be thrown and the report will fail to generate.

The application is now ready to run. Press **F5** to start the application and display the report. The report output should look similar to the following:



When the report loads, a listing of products by category is displayed along with the total products in stock for each category. Clicking on a category name, for example

Beverages, drills-down into the category, showing all of the products that make up the category and the units in stock for each product, as shown below.



The following list summarizes the steps taken in this section to create the sample report.

- Add a new Crystal Report object to the project.
- Manually layout the report using the Blank Expert
    - Connect to the database
    - Add the view to the report
    - Select columns from the view to appear on the report
    - Add a group header section to the report
    - Insert a subtotal to the group header section to the report
    - Drag and drop fields from the Field Explorer to the report's details section
    - Permit drilling down on the details section through the context menu by selecting the **Hide (Drill-Down OK)** option.
- Drag a ReportDocument component from the Toolbox and drop it on the web form in the project.
- Add code to the form's Page_Load event to bind the report to the viewer by setting the viewer's ReportSource property to the strongly typed ReportDocument object. The code should also set the database login credentials.

Scenario Conclusion

The "Intermediate" scenario illustrates the grouping and drill-down features available to a Crystal Reports report. The ReportDocument object, introduced in this scenario,

provides programmatic access to a report. This scenario used it to set the database login credentials and to bind the report to the report data. In future scenarios it will play a more prominent role in the report generation process.

The sample reports shown so far have been textual listings of data from the database. Neither report passed parameters to the queries to get their data. The “Visual” scenario builds on the previous two reports by adding graphical output and retrieving data by passing parameters to a stored procedure.

Visual Reporting Scenario

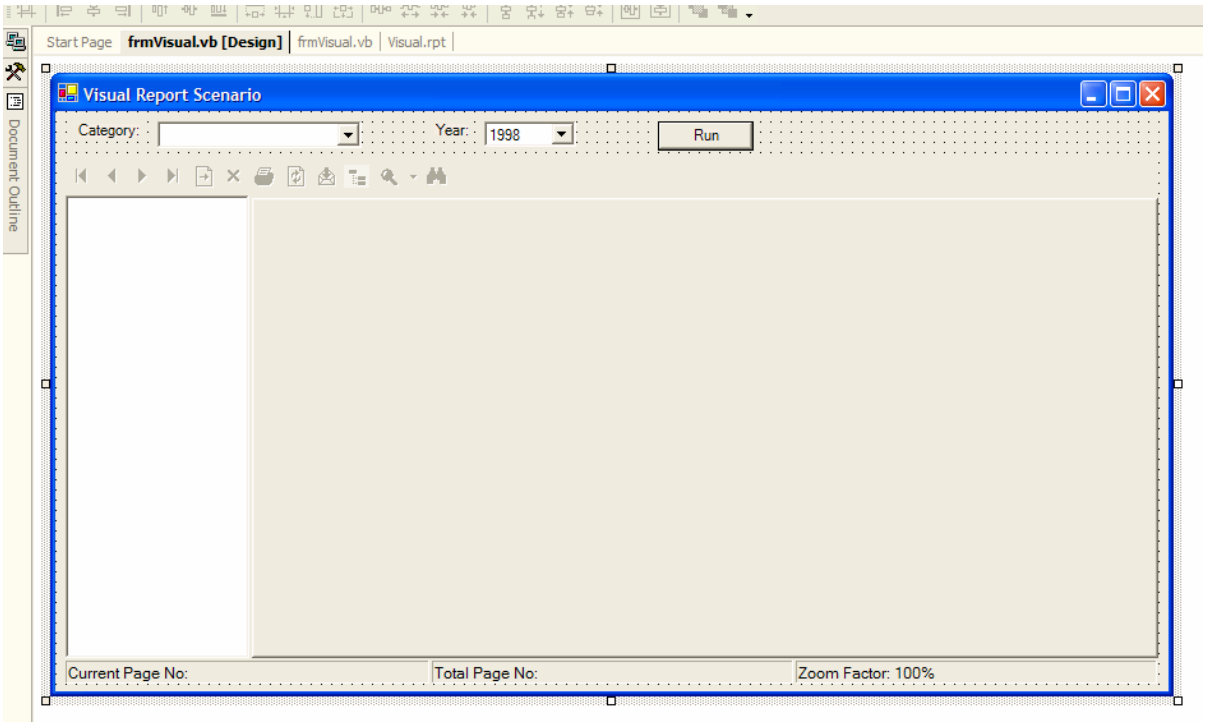
Charts make reports easier to read and provide a quick summary of information to the user. Not only is charting a means of presenting data — it is also an analysis tool. Users can drill-down on a chart, or a chart's legend, for detailed information. The Visual Reporting scenario adds charting components to the report.

This scenario also introduces the techniques for reporting off of a parameterized stored procedure. The *SalesByCategory* stored procedure accepts two parameters, category name and order year, and returns the total sales dollars for each product under the requested category.

The sample code for the Visual report can be found in the CRWindowsSamples application. The frmVisual form and Visual.rpt report demonstrate all of the techniques described here.

Report Layout

Add a new Windows form to the project and name it “frmVisual”. Add a ComboBox, DateTimePicker, Button, two Labels, and the CrystalReportViewer control to the form. Lay the controls out on the form as depicted in the figure below.



Set the following properties:

frmVisual

Property Name	Value
AcceptButton	btnRunReport
Text	Visual Report Scenario
WindowState	Maximized
Name	frmVisual

ComboBox

Property Name	Value
Name	cmbCategory
DropDownStyle	DropDownList

DateTimePicker

Property Name	Value
Name	cmbYear
CustomFormat	yyyy
Format	Custom
MaxDate	12/31/1998
MinDate	01/01/1996
Value	12/31/1998

Button

Property Name	Value
Name	btnRunReport
Text	Run

Label

Property Name	Value
Name	lblYear
Text	Year:
AutoSize	True

Label

Property Name	Value
Name	lblCategory
Text	Category:
AutoSize	True

CrystalReportViewer

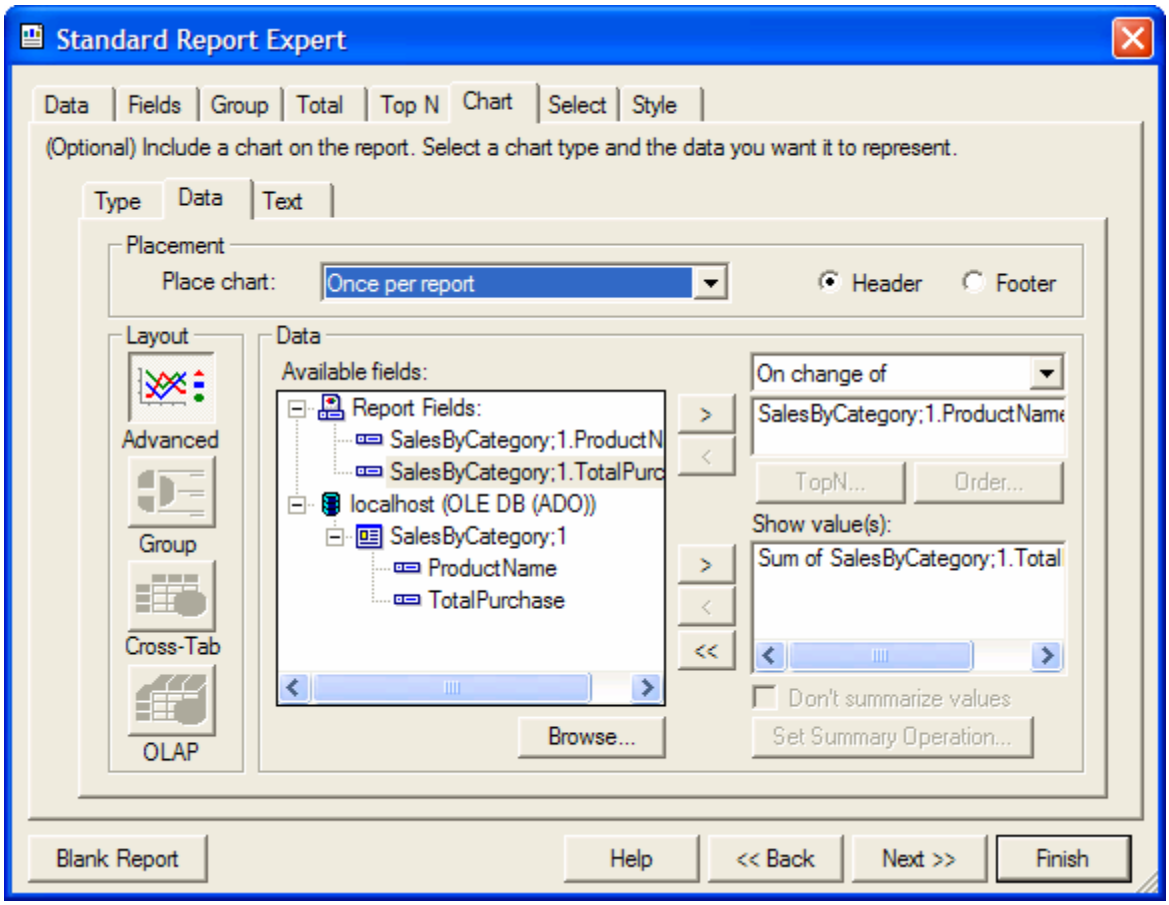
Property Name	Value
Anchor	Top, Bottom, Left, Right

After designing the form, add a new report object to the project by right-clicking on the project name in the Solution Explorer and choosing **Add | Add New Item…**. Select the Crystal Report object and name it "Visual.rpt". Click **OK** to start the Standard Expert.

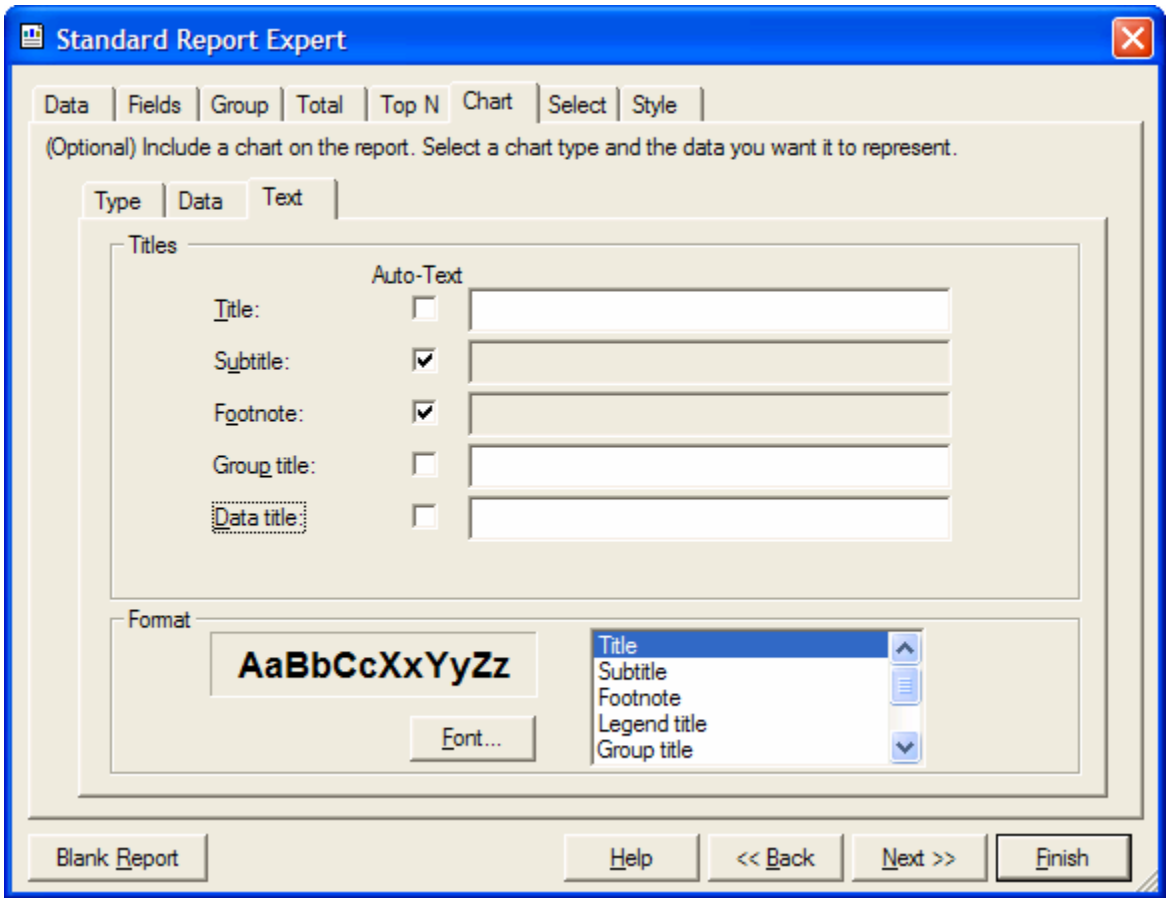
Review the section *Database Connection Used by Windows Based Reports* in appendix A to establish the connection to the database using a data link file. After establishing the connection to the database, select the *SalesByCategory* stored procedure and click the **Insert Table** button. Next, click the **Fields** tab in the Expert to add the fields from the stored procedure to the report. Click the **Add All ->** button. After selecting the fields, click on the **Chart** tab in the Expert.

Configuring a chart for the report is simplified with the Expert. This report uses a bar chart to display results. Select the **Bar** chart type and click on the button representing the 3D side-by-side chart layout, as shown below. Click the **Data** tab to setup the data for the chart axes.

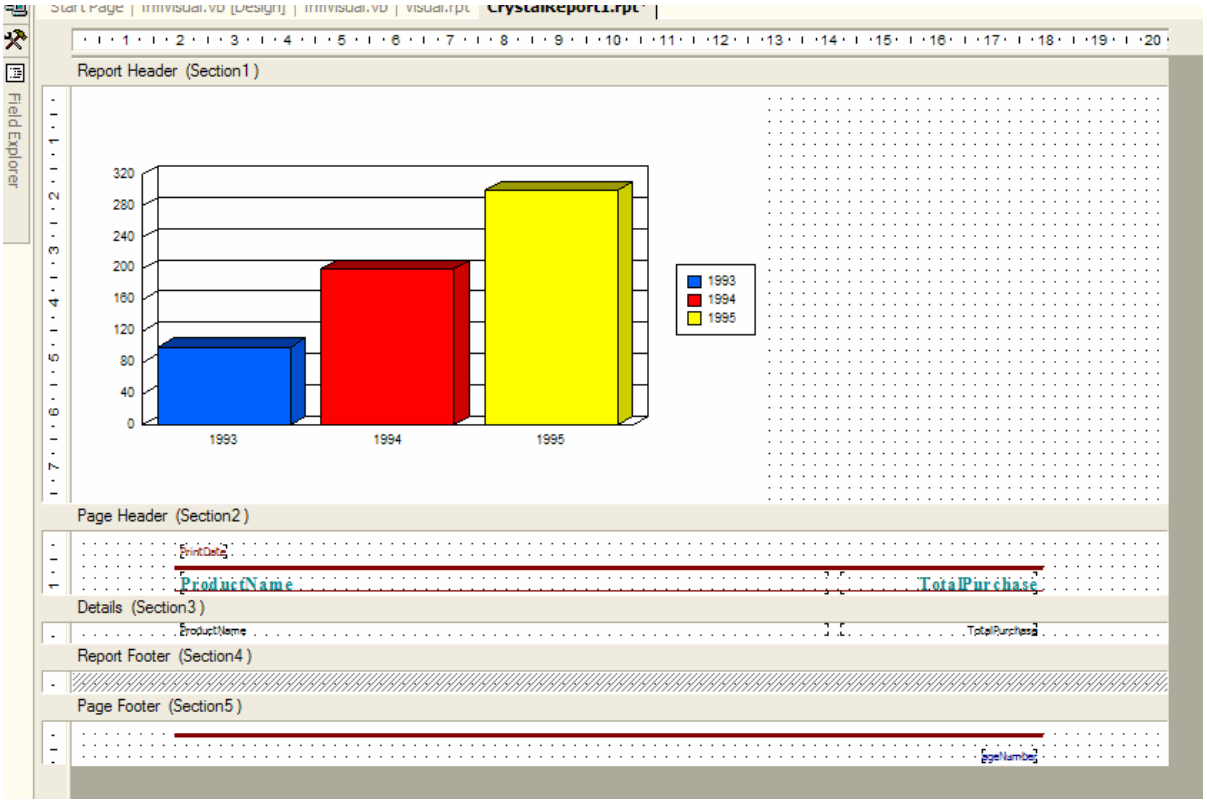
The data for the chart is grouped by product name. The total sales for the year for each product under a given category are displayed in the chart. Select the *SalesByCategory;1.ProductName* field from the list of available fields and add it to the list box under the *On Change of* drop down by clicking the button with the right arrow on it (**>**). Next, select the *SalesByCategory;1.TotalPurchase* field and add it to the *Show Value(s)* list box by clicking the second button with the right arrow on it **>**. After specifying the data, the form should appear as it does in the screen shot below.



The Visual report uses custom text for the group, data, and report titles. Click on the **Text** tab; uncheck the **Auto-Text** option for Title, Group title, and Data title. Remove any default values in those title textboxes. The final result should look like the following screen shot.



The last step is to set the report's style. Click on the Expert's **Style** tab, choose the *Maroon/Teal Box* option and click the **Finish** button. When viewed in the designer, the report should look similar to the one below.

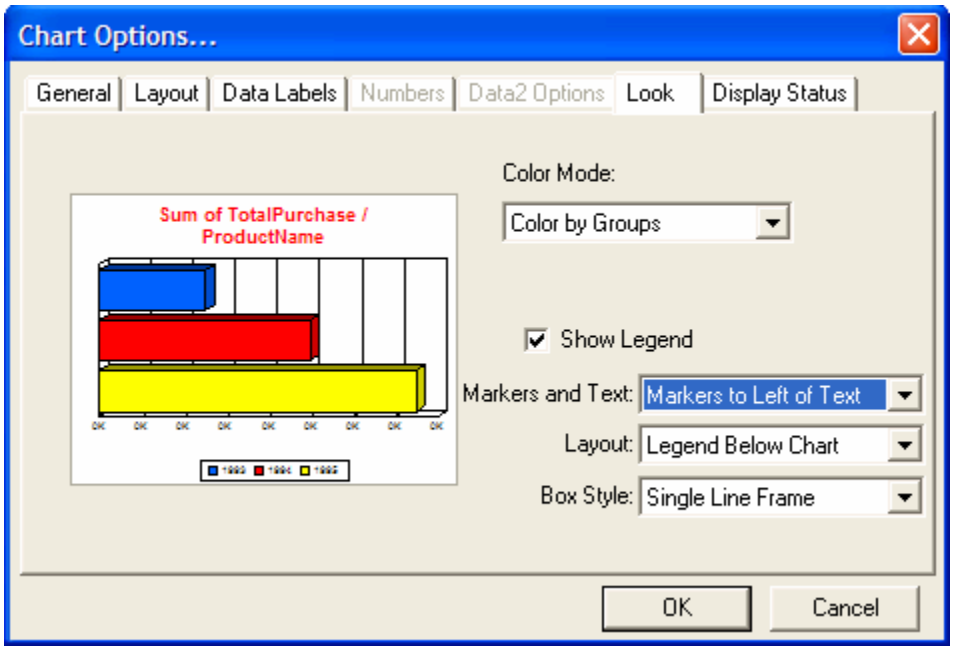


A few additional layout changes improve the effectiveness of the report. This report retrieves data based on criteria chosen by the user. The report is enhanced by displaying the chosen criteria in the report title.

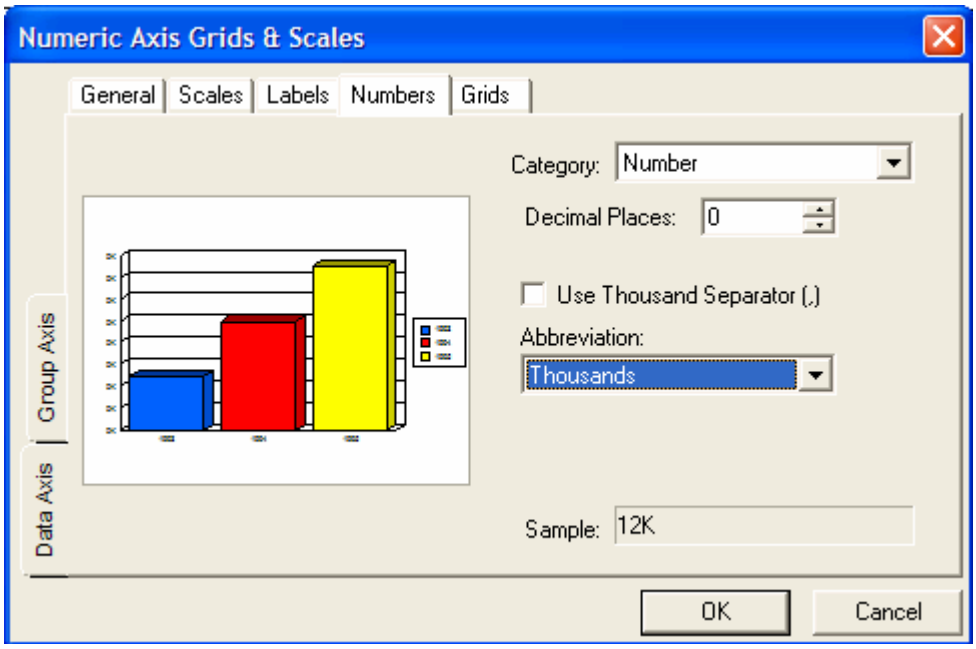
Drag the category name (*?@CategoryName*) and order year (*?@OrdYear*) parameters from the Fields Explorer, and drop them in the report header section just above the graph. Add a Text object between the two parameter fields and set the text to “Sales by Product for”. This will become the report header.

Increase the size of the graph to improve readability. Resize the graph so that it takes up the full width of the report.

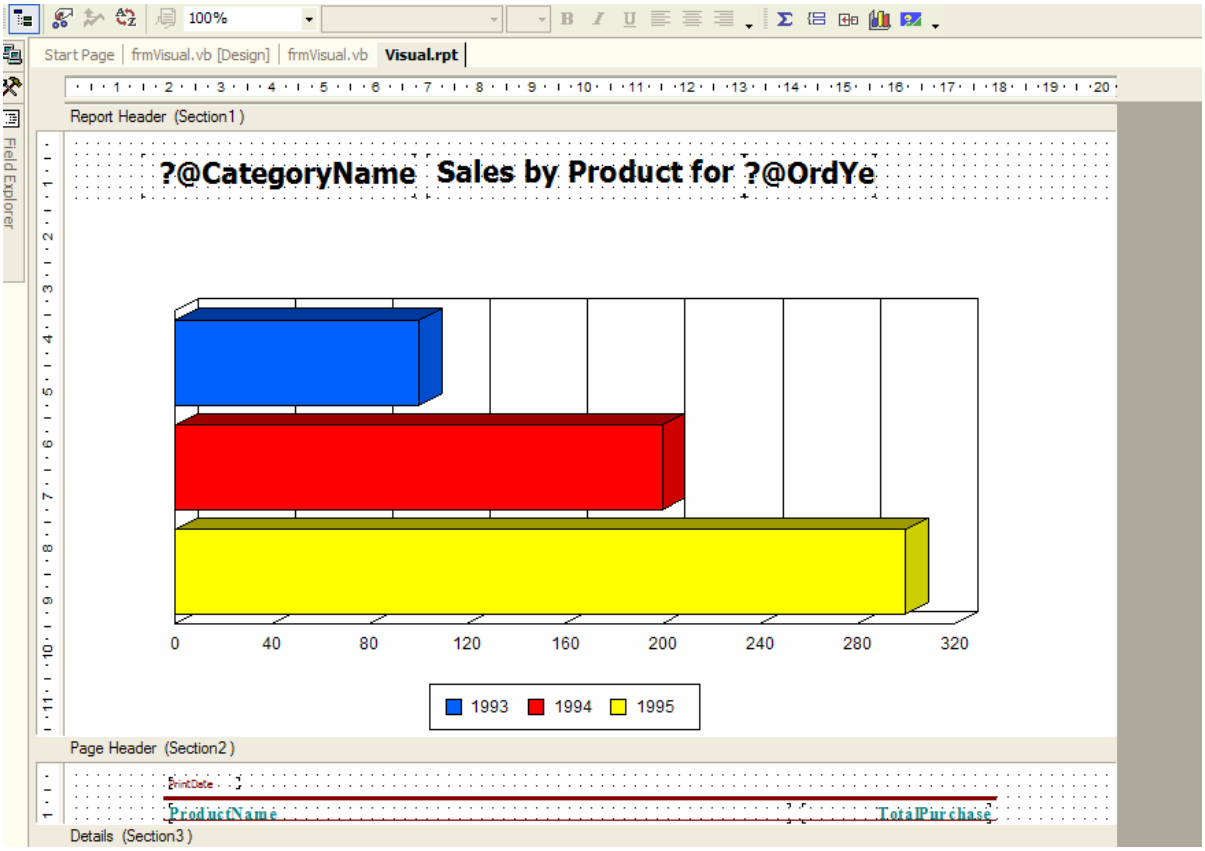
Change the location of the chart’s legend by right-clicking on the chart and choosing **Format Chart > | General...** from the menu. The Chart Options dialog loads. From the options, choose the **Look** tab. In the *Markers and Text* drop down, select the **Markers to Left of Text** item. In the *Layout* drop down, select the **Legend Below Chart** item. This places the legend below the chart, giving the chart more room to display data horizontally. Click **OK** to accept the layout changes.



The final layout change is to change the data labels. The labels use shorthand to represent values in the thousands (K). Right-click on the chart, and choose **Format Chart > | Grid...** from the menu. The following dialog appears.



Click on the ***Data Axis*** tab on the left hand side of the form. Next, click on the **Numbers** tab on the top of the form. Select the **Thousands** item from the *Abbreviation* drop down. Click **OK** to close the form and accept the changes. The report layout is complete and should look similar to the following.



Putting it Together

The completed report needs to be bound to the viewer. The sample report uses a ReportDocument object to perform the binding. Add a ReportDocument component to the frmVisual form by dragging it from the Toolbox and dropping it onto the form. Select the **CRWindowsSamples.Visual** typed report from the *Name:* drop down and click the **OK** button. An instance variable named “visual1” is added to the form. Report binding occurs using the visual1 instance variable.

In addition to binding the report, the ReportDocument exposes the **SetParameterValue** method. As its name implies, the SetParameterValue method passes parameter values to the report’s stored procedure.

Code in the btnRunReport’s click event sets the reports stored procedure parameters to values selected in the cmbCategory and cmbYear combo boxes. After setting the parameters, the viewer is bound to the visual1 ReportDocument, which generates and displays the report. The following code in the btnRunReport button’s click event sets the stored procedure parameters and binds the report.

```
Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
Cursor.Show()

Try
```

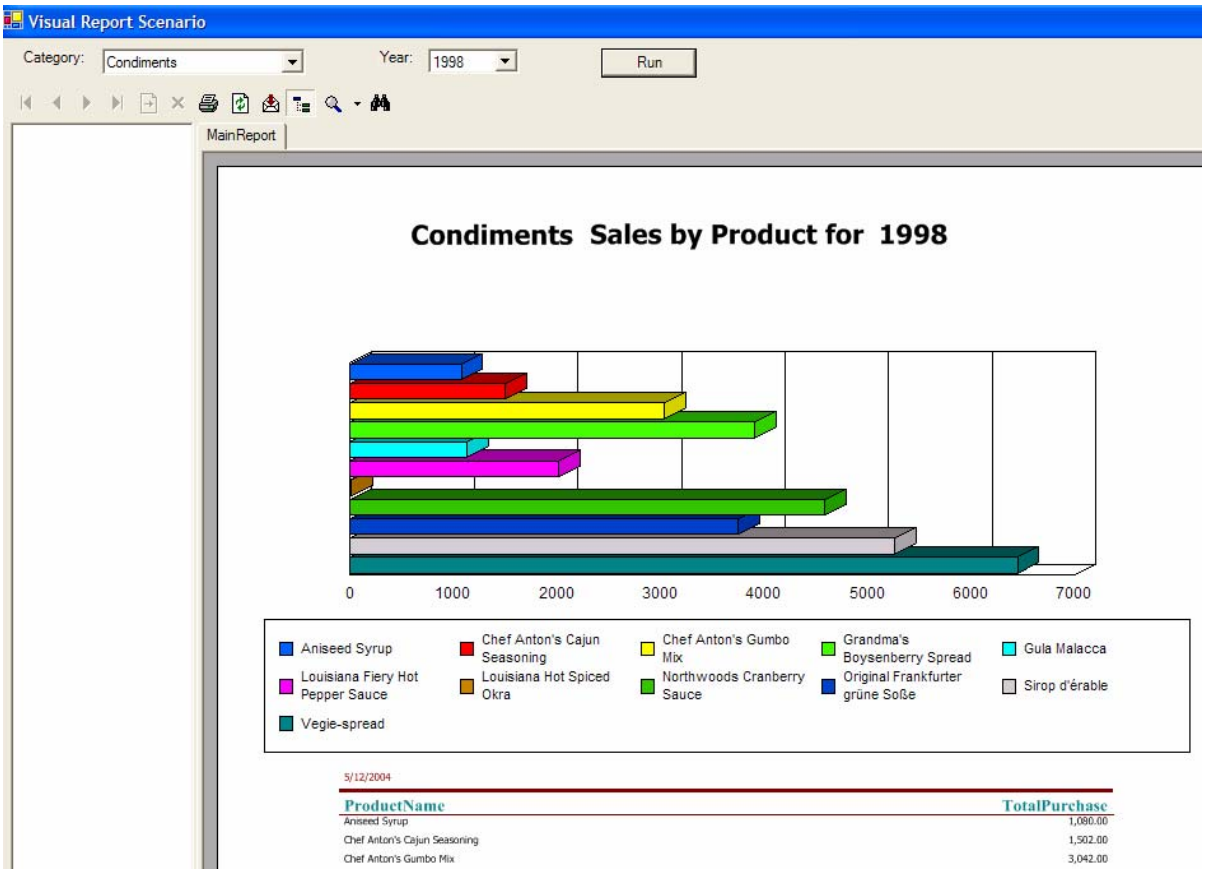
```
            ' set the report parameters
            With visual1
                .SetParameterValue("@CategoryName", _
                  cmbCategory.SelectedValue)
                .SetParameterValue("@OrdYear", cmbYear.Text)
            End With

            ' bind the report to the strongly typed
            ' ReportDocument (visual1) object
            CrystalReportViewer1.ReportSource = visual1

        Catch ex As Exception
            MessageBox.Show(ex.Message, _
              "Could not create report", MessageBoxButtons.OK)
        Finally
            Cursor.Current = System.Windows.Forms.Cursors.Default
            Cursor.Show()
        End Try
```

When the form loads, the cmbCategory drop down is populated from the Categories table in the Northwind database. Add the following code to frmVisual's load event to populate the drop down.

```
        ' connect to the database using
        ' Windows authentication.
        ' in a production environment, do not
        ' hard code the connection string,
        ' or use "inline" SQL statements.
        ' load the category drop down with
        ' category values from the database

        Dim da As New SqlDataAdapter _
          ("Select CategoryName from Categories", _
          "Data Source=localhost;initial catalog=northwind;" & _
          "Integrated Security=SSPI;Persist Security Info=False;")
        Dim ds As New DataSet("Categories")
```

```
    Try

        Cursor.Current = System.Windows.Forms.Cursors.WaitCursor
        Cursor.Show()
        da.Fill(ds)

        With cmbCategory
            .DataSource = ds.Tables(0)
            .DisplayMember = "CategoryName"
            .ValueMember = "CategoryName"
        End With

    Catch ex As Exception
        MessageBox.Show(ex.Message, _
          "Could not load categories", MessageBoxButtons.OK)
    Finally
        da.Dispose()
        Cursor.Current = System.Windows.Forms.Cursors.Default
        Cursor.Show()
    End Try
```

To see the report in action, set the project's startup object to frmVisual and run the application by pressing **F5**. Select the *Condiments* category and default year (1998) and click on the **Run** button to generate the report. The report should appear similar to the following.



The following list summarizes the steps taken in this section to create the sample report.

- Add a new Crystal Report object to the project.
- Layout the report using the Standard Expert
    - Connect to the database
    - Add stored procedure to the report
    - Select the columns from the table to appear on the report
    - Add a chart to the report
        - Specify the data to display in the chart
        - Format the chart
    - Set the report style
- Add a CrystalReportViewer control to the windows form in the project.
- Bind the report to the viewer through code.
    - Gather stored procedure parameters data from the form and apply them to the report's stored procedure using the ReportDocument's SetParameterValue method.
    - Bind the report to the viewer using the ReportSource property.

Scenario Conclusion

The Visual reporting scenario builds on the Intermediate scenario by adding charting and reporting on parameterized stored procedures. Colorful charts permit users to make snap decisions from report data. The Crystal Report Experts quickly add and configure

charts for any report. Reporting based on parameterized stored procedures is facilitated through a strongly typed ReportDocument's SetParameterValues method.

While charting is a great way to enrich a report, the next scenario, Sub-report, further enhances reporting by embedding a report within a report.

Sub-report Reporting Scenario

Two separate reports can be combined into one using the sub-report features provided by Crystal Reports. A sub-report is a report embedded within a report. A sub-report is not required to be related to the container report (main report).

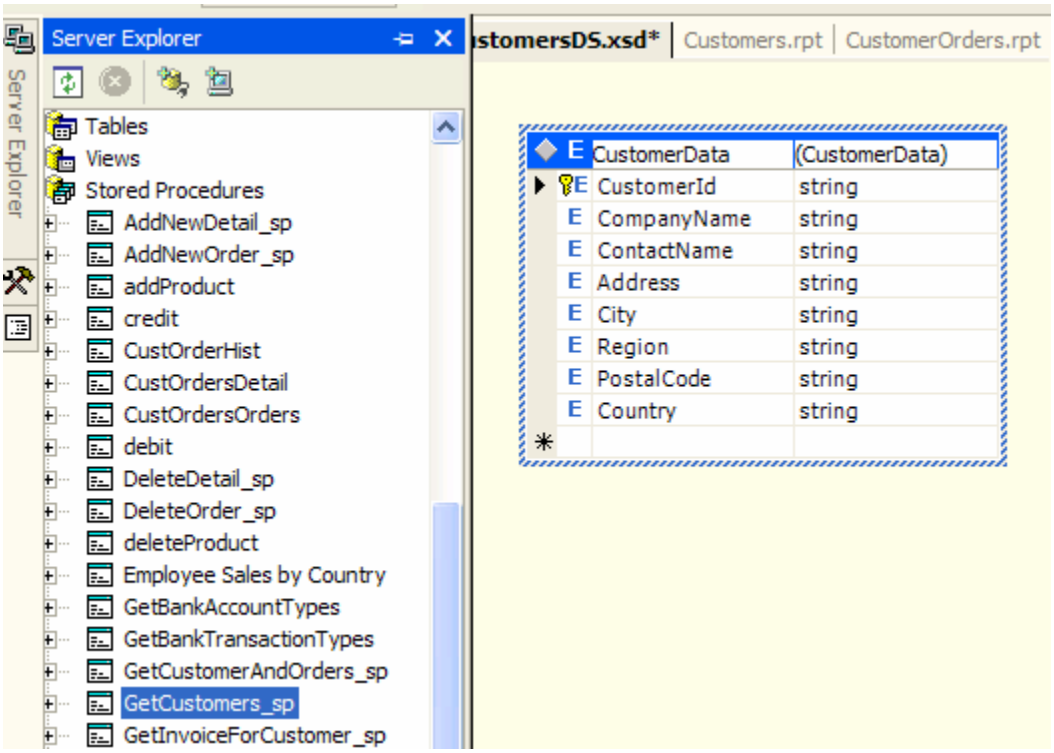
Sub-reports can be unlinked or linked. Unlinked sub-reports are not related to the main report in any way. Linked reports are related to the main report. When linking reports, a parameter field is used to coordinate the data between the two reports. The sub-report uses the parameter in a selection formula to display only the data associated with the main report's link field. Performance can be a concern when linking a sub-report to a main report because a sub-report is created for each record in the main report.

Performance suffers if the sub-report contains large amounts of data. To prevent performance degradation, a sub-report can be configured to run "on-demand". On-demand sub-reports are displayed as hyperlinks in the main reports. Sub-report data is only retrieved when a user clicks on the hyperlink.

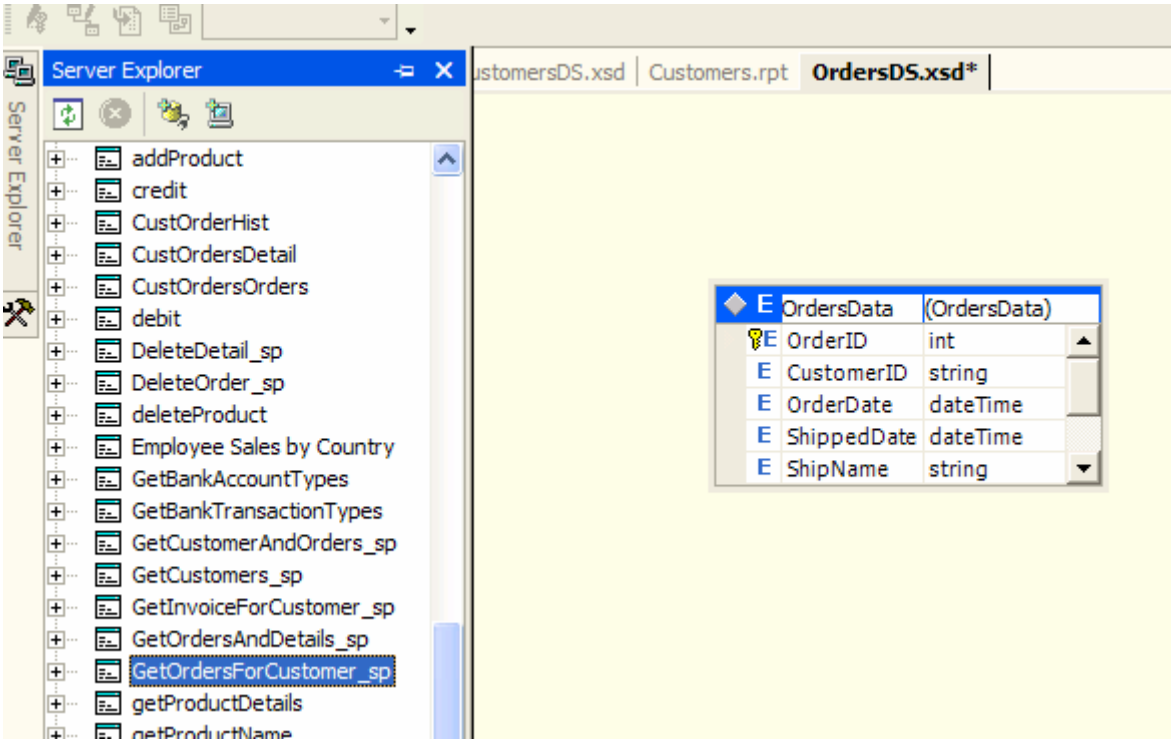
This scenario expands on the customer listing report created in the Basic scenario by linking the report to a sub-report of orders. Sub-reports are displayed on-demand, using a hyperlink. The reports are created as standalone report files and deployed as part of a Web application. Data for the sub-report is collected using an ADO.NET typed DataSet.

The sample report uses the *GetCustomers_sp* stored procedure to retrieve data for the main report (Customers.rpt) and the *GetOrdersForCustomer_sp* to retrieve data for the sub-report (Orders.rpt).

The report and sub-report in this scenario get their data from typed ADO.NET DataSets. Create two typed DataSets; one for customer data and one for order data. The process to create a typed DataSet is shown in the Visual scenario section above. The customer DataSet is based on the GetCustomers_sp stored procedure. After adding the stored procedure to the DataSet designer, name the DataSet "CustomersDS" and name the DataTable "CustomerData". The CustomersDS DataSet should look like the DataSet depicted below.



The order DataSet is based on the GetOrdersForCustomer_sp. Again, after adding the stored procedure to the designer, name the DataTable “OrdersData”. The OrdersDS DataSet should look like the DataSet depicted below.



The next step is to create the sub-report, *CustomerOrders.rpt.* Add a new Crystal Report object to the project and name it “CustomerOrders.rpt”. This report will show the orders

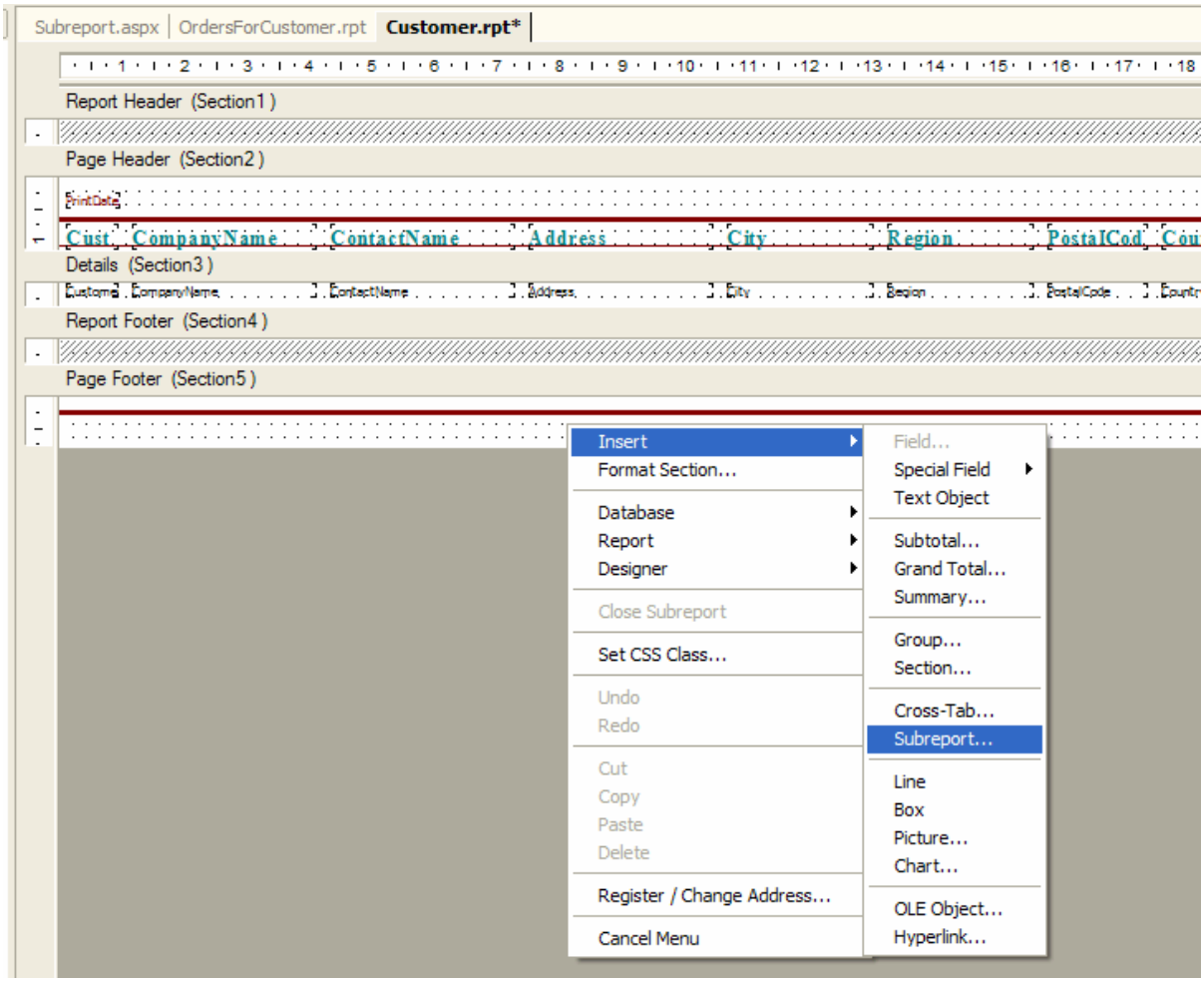
associated with a given customer. Using the Standard Expert, select the **OrderData** DataTable from the list of available data sources. The OrderData DataTable can be found by expanding the *Project Data* | *ADO.NET DataSets* | *CRWebSamples.OrdersDS* nodes.

Select all available fields to display on the report from the Expert. Set the report style to *Maroon/Teal Box*.

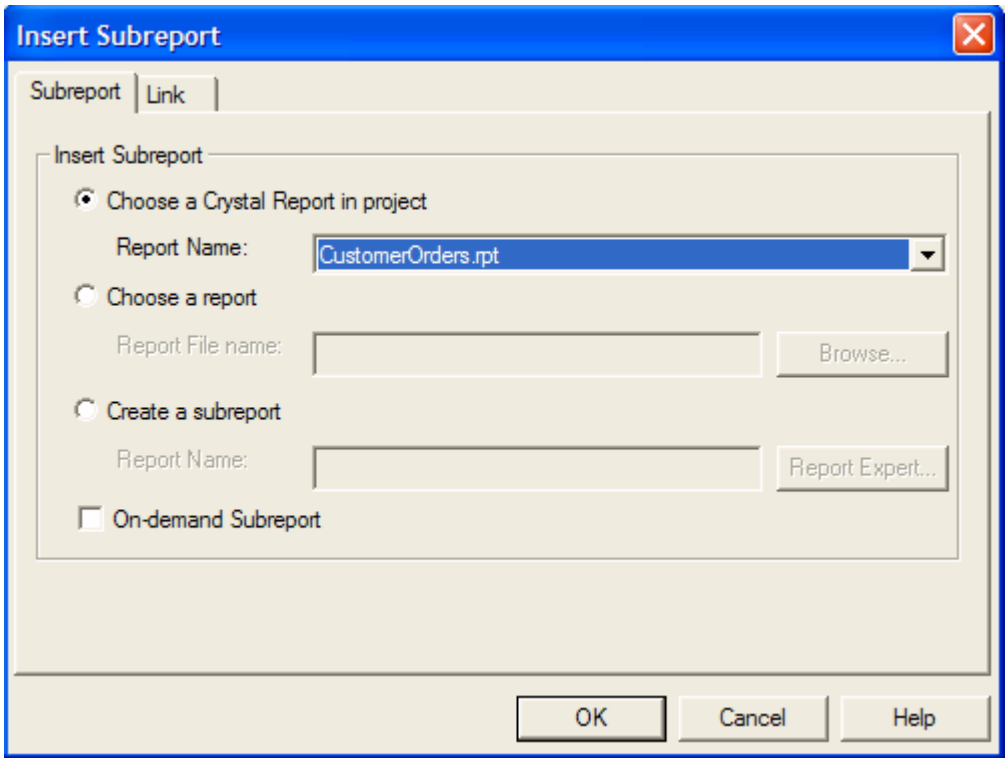
Next, create the main report. Add a new report to the project and name it "Customers.rpt". This report lists the customers in the customer table and will contain a linked sub-report. Use the Standard Expert to select the **CustomerData** DataTable, all fields, and to set the style to *Maroon/Teal Box*. Close the Expert to view the report in the design window.

Report Layout

The main report should look similar to the one pictured below. The next step is to embed the sub-report within the main report. Right-click on the report and choose **Insert | Subreport...**. Drop the sub-report object into the Details section of the report. Expand the height of the Details section if necessary.

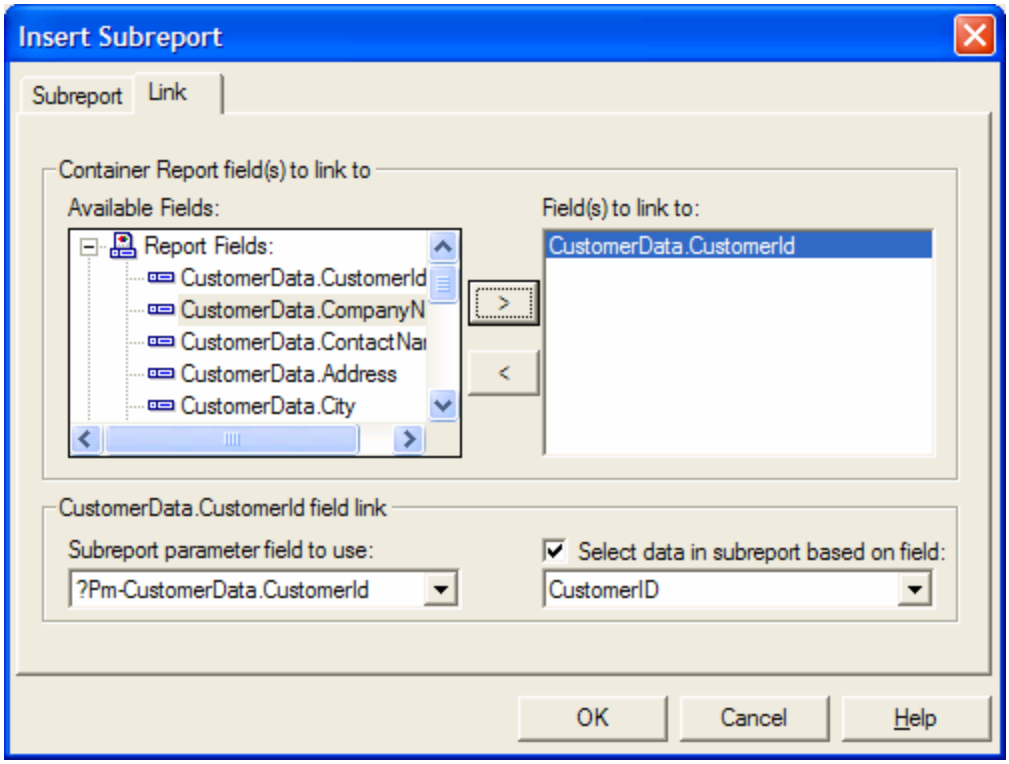


A dialog window opens to guide the process of adding the sub-report. The source of the sub-report can be an existing report in the project, a report file, or a new report. Choose the **Choose a Crystal Report in Project** option and select the **CustomerOrders.rpt** from the drop down list. This main report use on-demand linking. Place a check in the **On-demand Subreport** checkbox to set up on-demand linking.

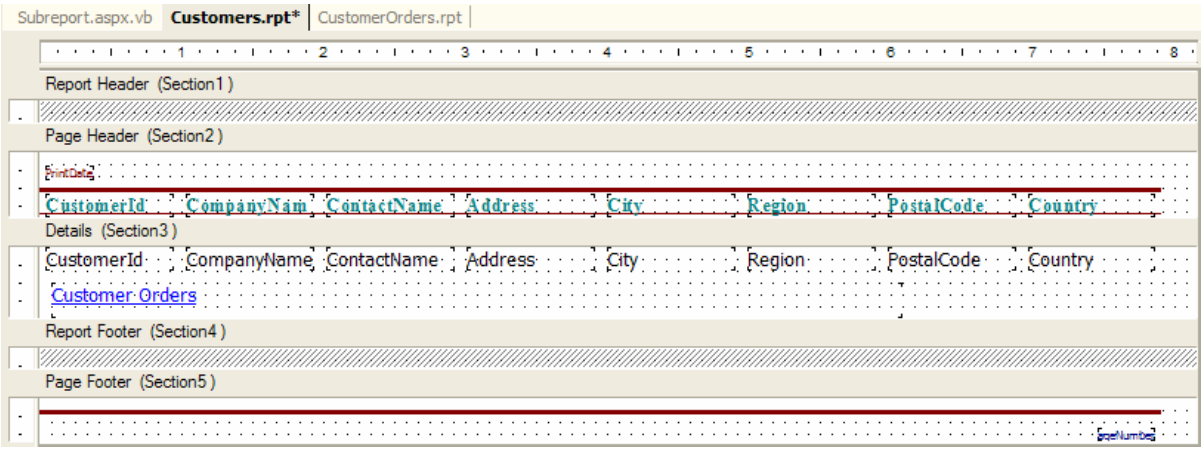


Next, click the **Link** tab. The Link tab provides the details of how the two reports are related to one another. The reports used in this scenario are related by CustomerID. From the Available Fields list, select the *CustomerData.CustomerID* field and click the **>** button to add the field as a link field.

Report linking uses a parameter field to associate the sub-report with the main report. From the *Subreport parameter field to use* drop down, select the **?Pm-CustomerData.CustomerId** value. Check the **Select data in subreport based on field** checkbox and select the **CustomerID** value from the drop down. The dialog window should look like the following screen shot.



Click **OK** to accept the values and close the dialog. The following screen shot shows how the customers report should look after the sub-report has been embedded within it.

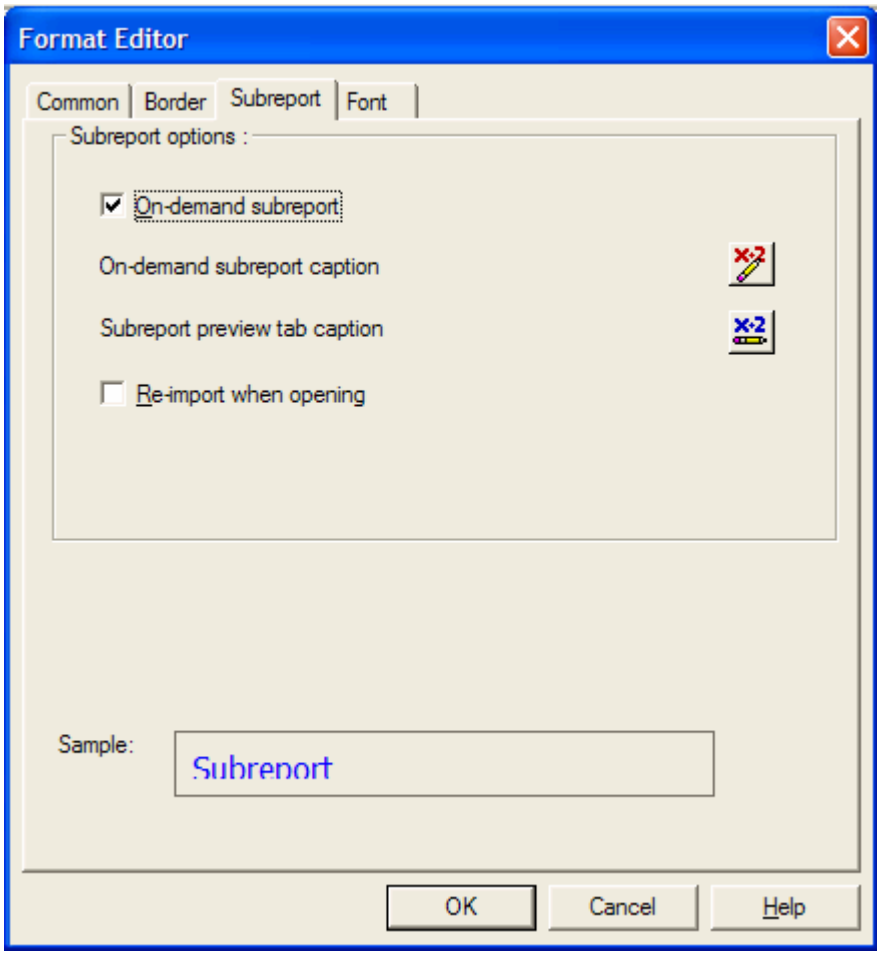


To make the report more presentable, increase the font size of the database field objects to a comfortable size. Change the following properties of the sub-report.

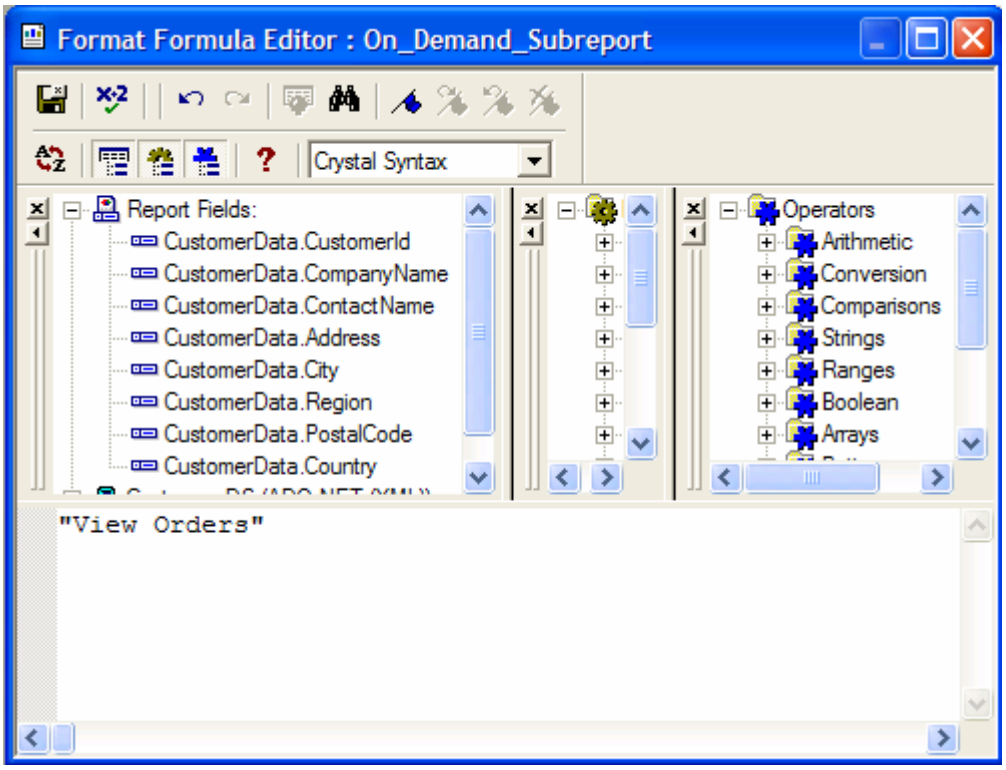
Property Name	Value
Name	CustomerOrders
Subreportname	Customer Orders

By default the hyperlink text used to display the on-demand sub-report is the report name. Luckily, this text can be changed to something more meaningful. To change the

hyperlink text, right-click on the sub-report object embedded in the main report and select the **Format…** option from the menu. Click on the **Subreport** tab as shown below.



Click on the **Formula** button to the right of the *On-demand subreport caption* option. This opens the Format Formula Editor dialog. For this sample, the hyperlink text will be the static text “View Orders”. Enter “View Orders” (quotes included) in the formula editor textbox on the bottom of the form, as shown below. Click the save button to save the report formula.



The reports are set up and ready to run. The next step is to set up a Web form that will contain the report. Add a new Web form to the project and name it “Subreport.aspx”. Add a CrystalReportViewer control and a ReportDocument component to the form. Set up the ReportDocument component so that it is based on the customers typed report.

Putting it Together

With the reports and form created, the final steps involve getting the data and binding the reports to the data. Code in the Web form’s Page_Load event fills two typed ADO.NET DataSets. These datasets provide the data for the main and sub-report.

After the DataSets are filled, they are assigned as data sources to the main report and sub-report ReportDocuments using the SetDataSource method. The main report ReportDocument object is the one added to the form at design time, Customers1. The ReportDocument exposes the ReportObjects collection from its ReportDefinition property. The ReportObjects collection contains all of the objects in the report, from text objects, to field objects. One of the items in the collection is the Subreport object embedded in the report. The sub-report object is retrieved from the collection and assigned to a Subreport Object with the following line of code.

```
subreportObject = customers1.ReportDefinition. _
  ReportObjects.Item("CustomerOrders")
```

Next, the sub-report object’s name is retrieved from the SubreportObject and assigned to a local variable.

```
subreportName = subreportObject.SubreportName
```

The code then calls the main (customers1) ReportDocument's OpenSubreport method. The OpenSubreport method opens the sub-report embedded in the main report and returns a ReportDocument Object corresponding to the named sub-report. The ReportDocument object returned is assigned to the ReportDocument variable "CustomerOrders".

```
CustomerOrders = customers1.OpenSubreport(subreportName)
```

The CustomerOrders ReportDocument represents the sub-report embedded in the main report. Set its data source to the typed disorders DataSet.

```
CustomerOrders.SetDataSource(dsOrders)
```

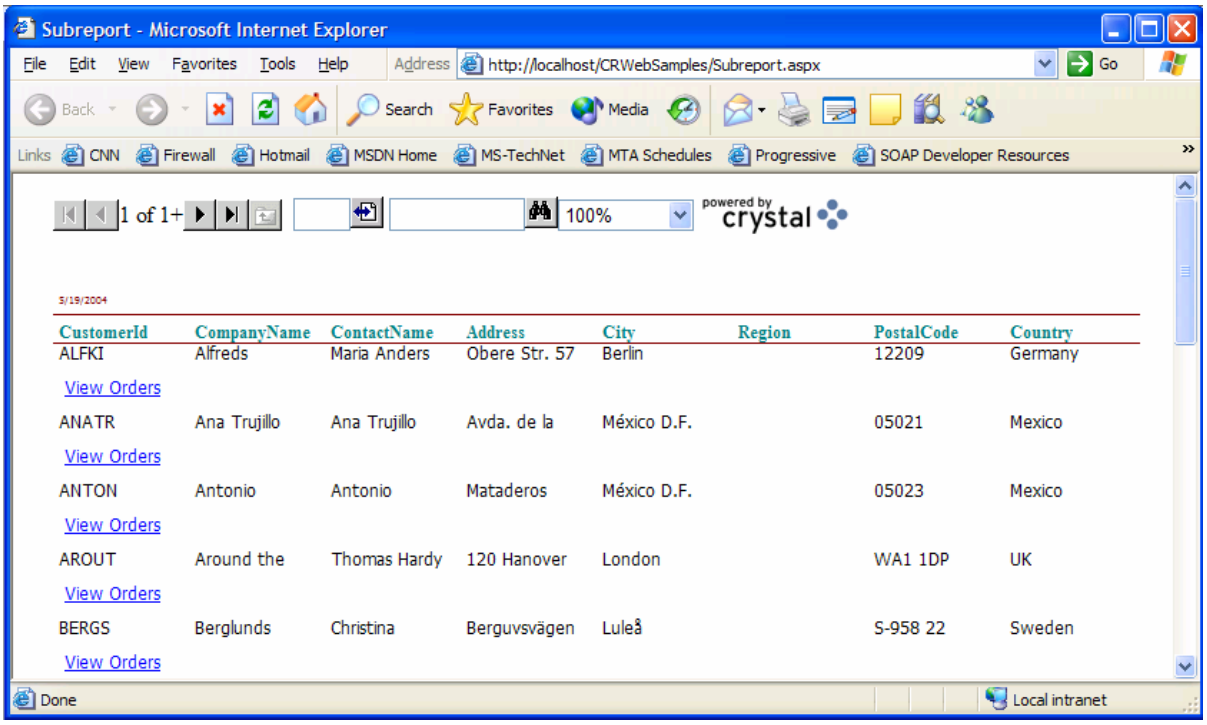
The report is now ready to run. To run the report, set the SubReport.aspx page as the startup page by right-clicking on the file in the Solution Explorer window and selecting the **Set as Start Page** option from the menu. The complete code listing for the Page_Load event is shown below.

```
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

    ' get all of the data for the report and sub-report
    Const db_connection As String = _
      "data source=localhost;initial catalog=northwind;" _
      & "user id=crreports;password=crreports;"

    Dim daNorthwind As New SqlClient.SqlDataAdapter( _
      "GetCustomers_sp", db_connection)
```

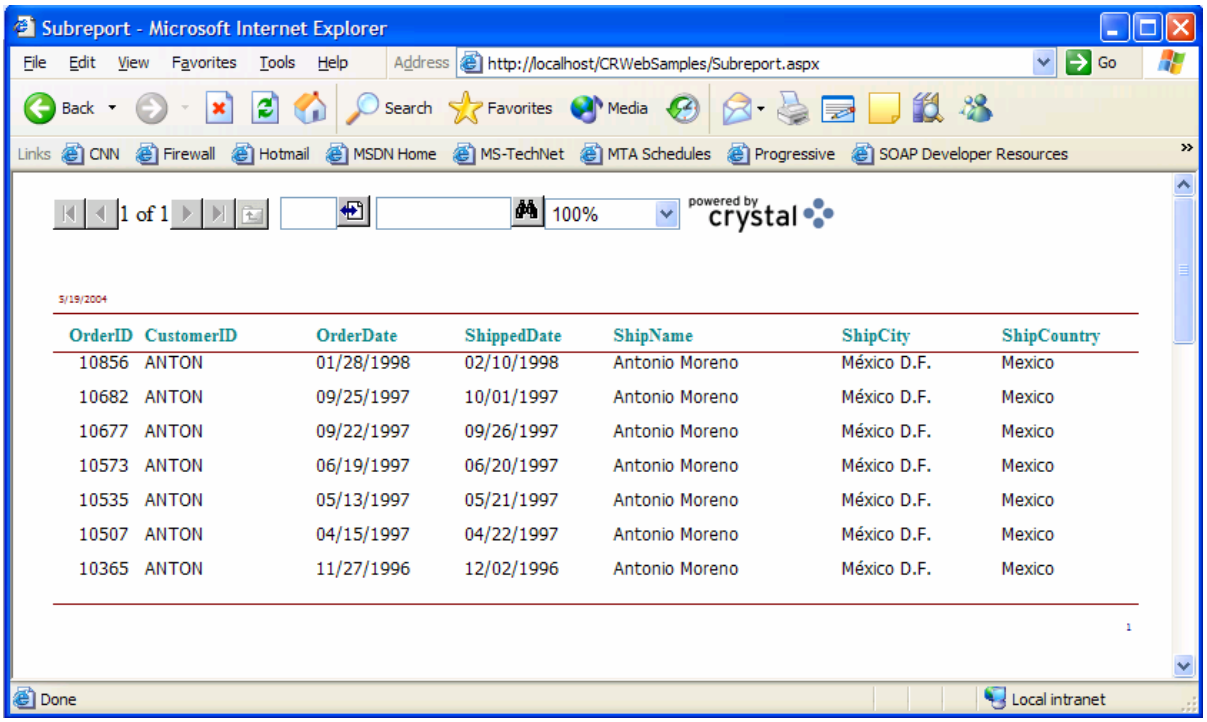
```
        Dim dsCust As New CustomersDS
        Dim dsOrders As New OrdersDS

        ' use the same adapter to get both datasets
        With daNorthwind

            .SelectCommand.CommandType = _
              CommandType.StoredProcedure

            ' map the generic "Table" to the typed
            ' datatable named CustomerData"
            .TableMappings.Add("Table", "CustomerData")

            ' fill the customer dataset
            .Fill(dsCust)

            .TableMappings.Clear()
            .SelectCommand.CommandText = _
              "GetOrdersForCustomer_sp"

            ' re-map the generic "Table" to the typed
            ' datatable named OrdersData"
            .TableMappings.Add("Table", "OrdersData")

            ' fill the order dataset
            .Fill(dsOrders)

        End With

        ' get the sub-report object from
        ' the main Customers1 ReportDocument
        Dim subreportName As String
        Dim subreportObject As SubreportObject
        Dim CustomerOrders As New ReportDocument

        ' Get the ReportObject by name as a
        ' SubreportObject.
        subreportObject = customers1.ReportDefinition. _
```

```
          ReportObjects.Item("CustomerOrders")

        ' Get the sub-report name.
        subreportName = subreportObject.SubreportName

        ' Open the sub-report as a ReportDocument.
        CustomerOrders = customers1.OpenSubreport(subreportName)

        ' set the subreport's data source
        ' to the orders typed dataset
        CustomerOrders.SetDataSource(dsOrders)

        ' set the main report document's data source
        ' to the customers typed dataset
        customers1.SetDataSource(dsCust)

        CrystalReportViewer1.ReportSource = customers1

        dsOrders.Dispose()
        dsCust.Dispose()
        daNorthwind.Dispose()

    End Sub
```

When the report runs, the user is first presented with a list of customers. Underneath each customer is a hyperlink labeled *View Orders*.



If the user clicks on the hyperlink, the *CustomerOrders* sub-report loads and is displayed in the browser.



The following list summarizes the steps taken in this section to create the sample report.

- Create two strongly typed DataSets and add them to the project
    - Drop a stored procedure on the DataSet at design time from the Server Explorer
- Add a new Crystal Report object to the project for the sub-report report

- Add a new Crystal Report object to the project for the main report
- Use the Standard Expert to format each report and set the appropriate data sources
- Add a new web form to the project
    - Add the CrystalReportViewer control to the form
    - Drag a ReportDocument component from the Toolbox and drop it on the web form. Set its typed ReportDocument class property to the proper typed report document object
    - Add code to the form's Page_Load event to retrieve data using the typed DataSets
    - Add code to get a ReportDocument object, representing the embedded sub-report, from the main ReportDocument object
    - Add code to the form's Page_Load event to set the sub-report and main report data sources to the respective DataSets

Scenario Conclusion

This scenario described the steps to take to embed a report within another. Robust reporting solutions can be created by combining two separate reports into one. Crystal Report's sub-report feature embeds a sub-report within a container report seamlessly. Sub-reports can be linked or unlinked. For performance considerations, linked reports should be configured to run on-demand.

The scenario also explained how to use ADO.NET typed datasets as report data sources.

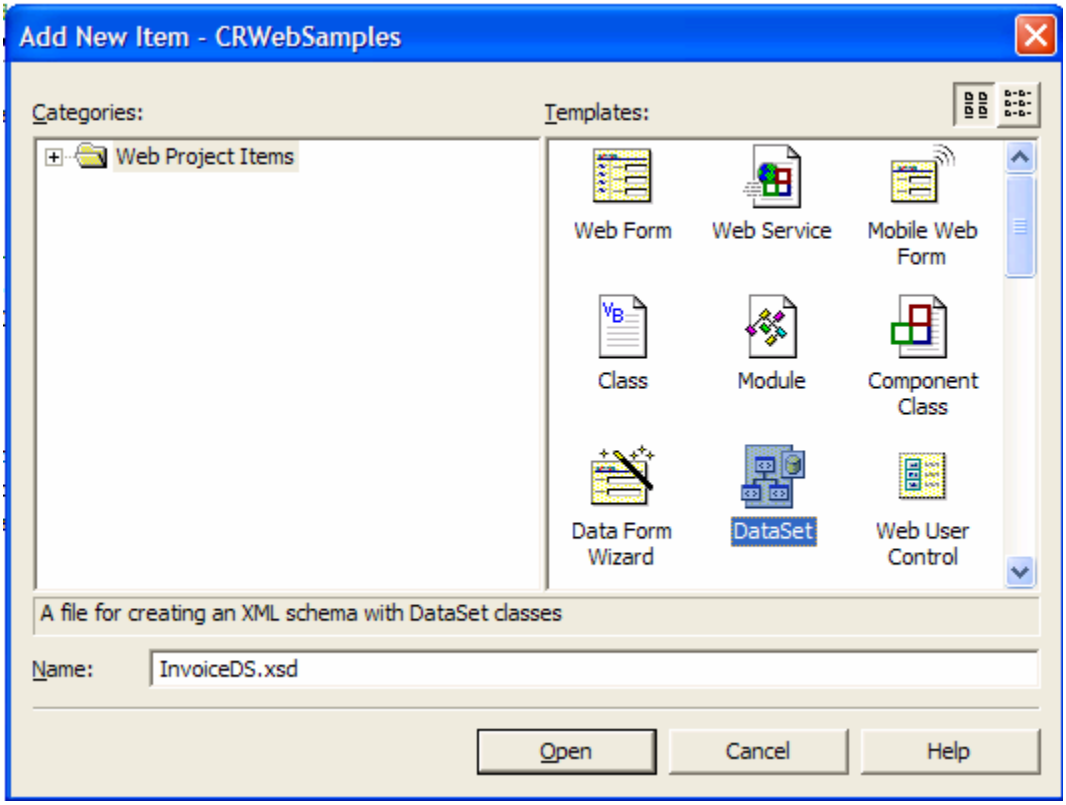
The next scenario continues to generate reports based on ADO.NET typed datasets. It also describes how to export reports to PDF with Crystal Reports for Visual Studio.NET 2003.

Document Reporting Scenario

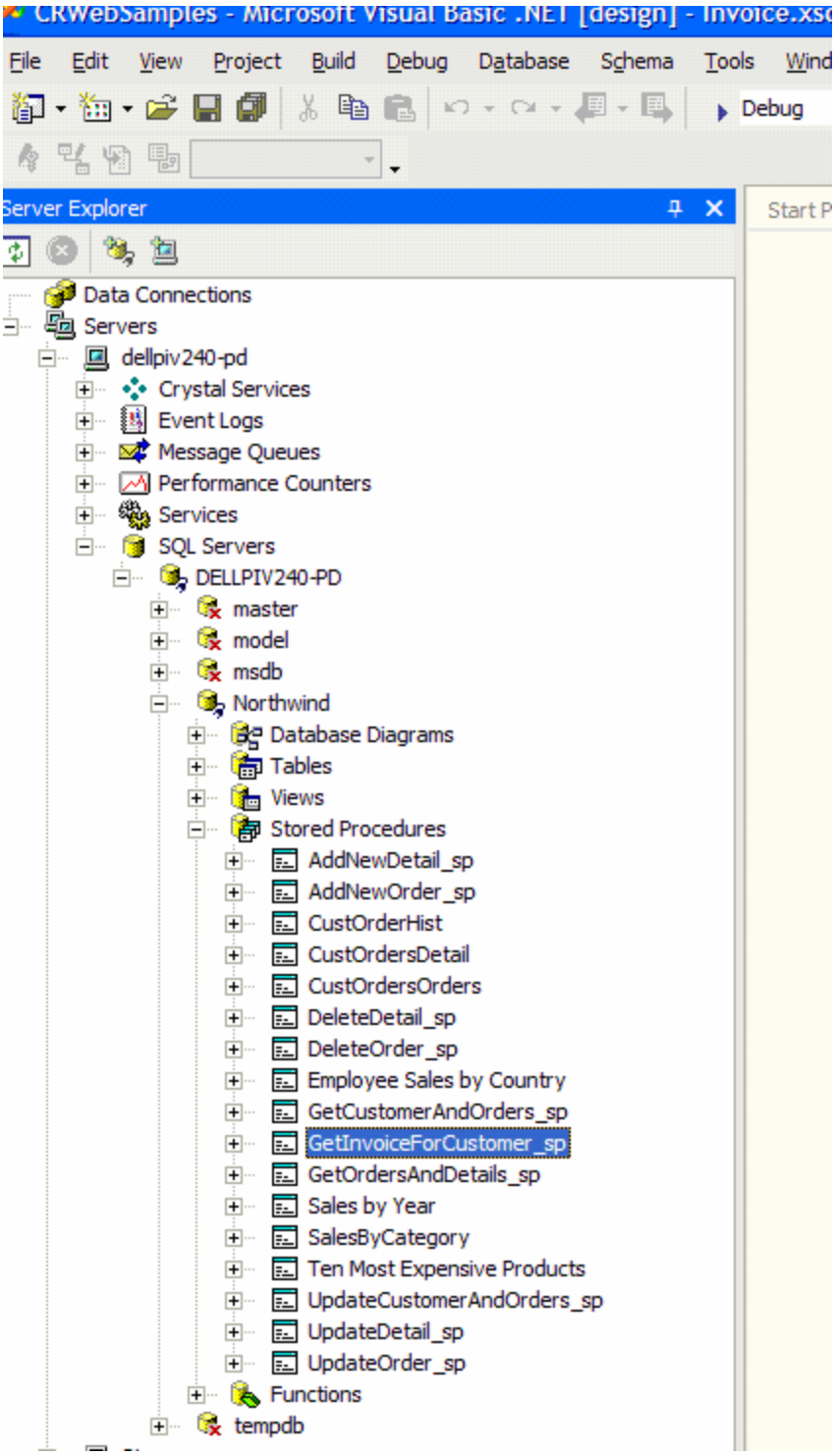
The Legal Document scenario illustrates how Crystal Reports can be used to generate legal documents, such as contracts or invoices, which are viewable in PDF format. The sample web form and report can be found in the CRWebSamples project. This report is created without the help of the Crystal Experts.

This scenario differs from the previous ones because it does not use the CrystalReportViewer control. Rather, the report is exported directly to PDF where it is streamed to the Web browser and displayed in Internet Explorer. A scenario such as this has all sorts of useful implementations, including business to business applications where legal documents are binding contracts. The exporting features provided by Crystal Reports are highlighted by the sample invoice document created here. The sample code retrieves data using the custom parameterized stored procedure, *GetInvoiceForCustomer_sp*

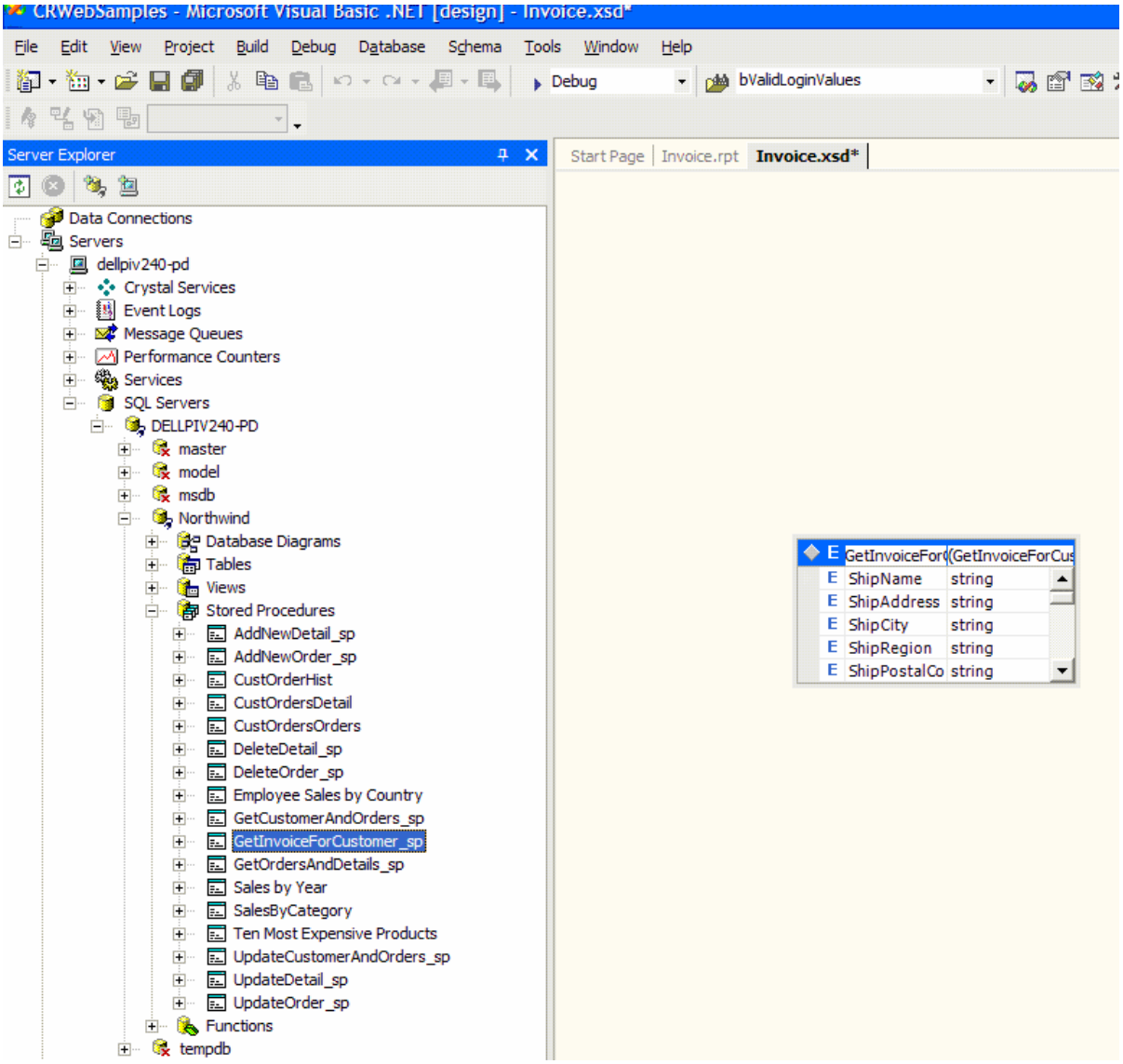
Next, create a typed dataset based on the GetInvoiceForCustomer_sp. Add a dataset to the project named "Invoice.xsd" by right-clicking on the CRWebSamples project in the Solution Explorer. Choose **Add | Add New Item…** and select the DataSet object from the ensuing dialog. Name the DataSet, "InvoiceDS.xsd" as shown below.



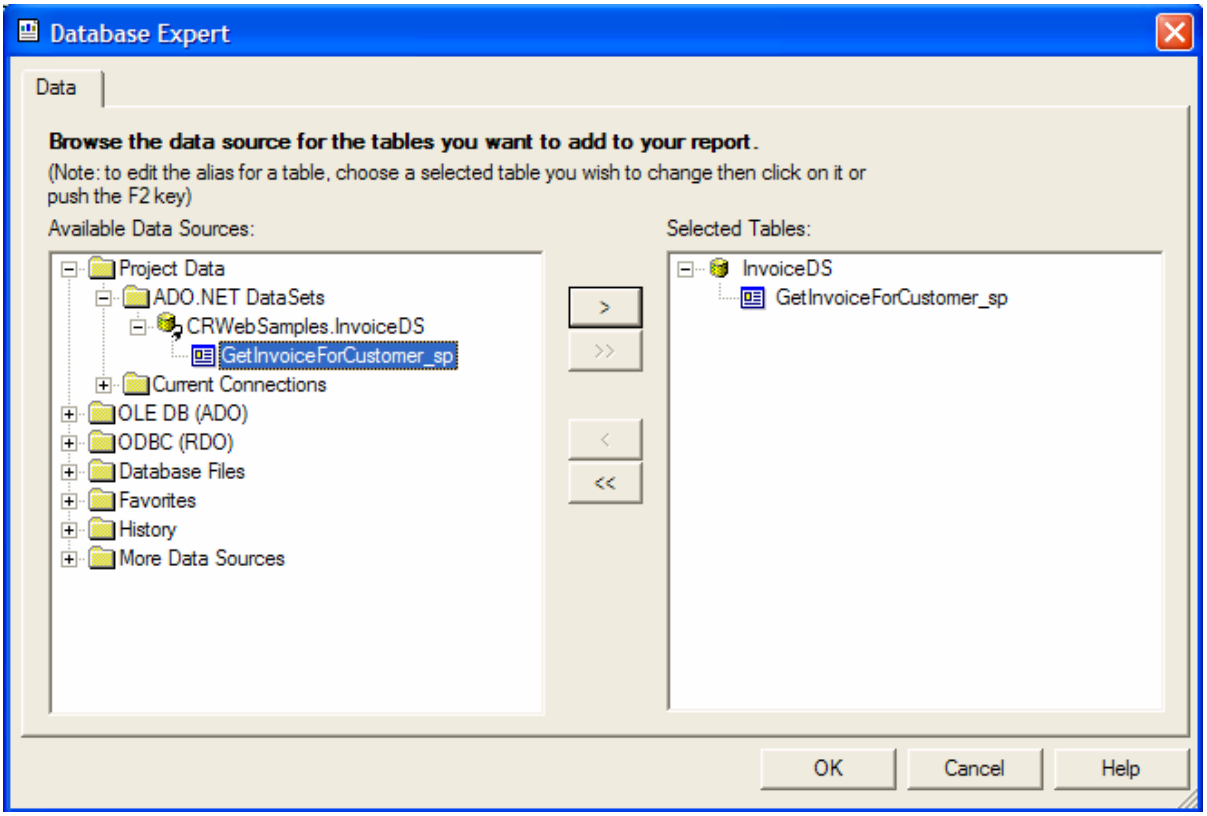
Click **Open** to add the DataSet to the project. The InvioceDS opens in design mode. Open the Server Explorer window. Starting with the *Servers* node, drill down to the *GetInvoiceForCustomer_sp* node (as shown below).



After locating the GetInvoiceForCustomer_sp node, drag it from the Server Explorer window and drop it onto the InvoiceDS DataSet. Dropping the stored procedure onto the DataSet creates a typed DataSet whose schema is defined by the columns returned by GetInvoiceForCustomer_sp. The DataSet should be similar to the following screen shot when viewed in design view.



Add a new report to the project named InvoiceDoc.rpt. When prompted choose the Blank Expert. The data source for this report is the typed DataSet, InvoiceDS created above. Right-click on the report and choose **Database | Add/Remove Database...** from the context menu. From the list of available data sources, expand the *ADO.NET DataSets* node, and then expand the *CRWebSamples.InvoiceDS* node. Select the **GetInvoicesForCustomer_sp** node and click the **>** button to add it to the list of selected tables. The following screen shot shows the results of this operation.

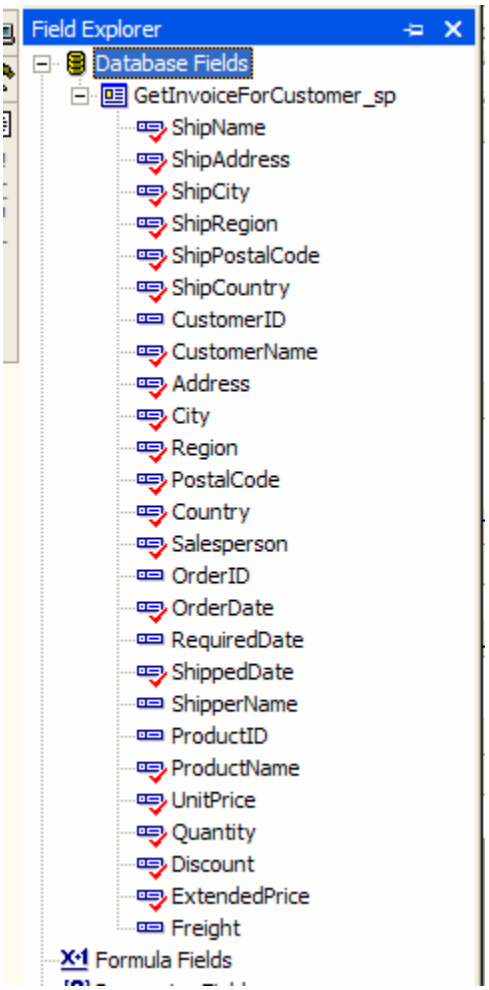


Report Layout

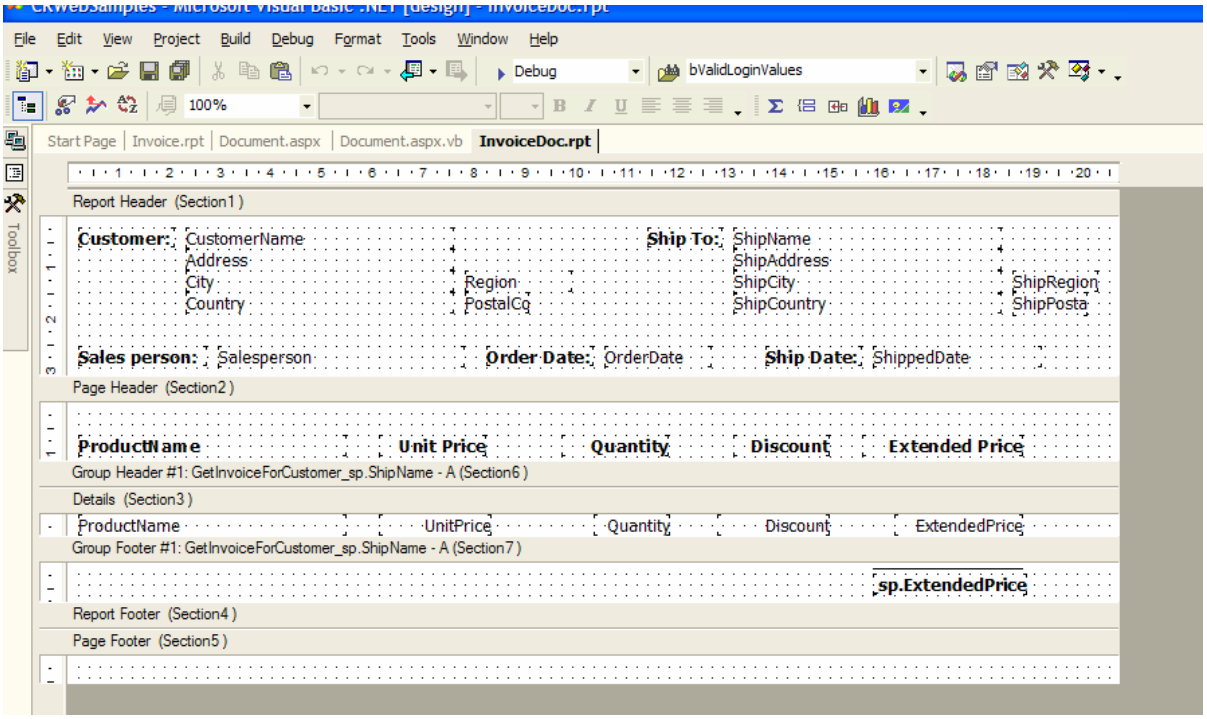
Click **OK** to close the Database Expert dialog. Drag the following fields from the Field Explorer window onto the report. Fields added to the report will be displayed with a red check to the left of their names in the Field Explorer, as shown below.

- Add the following fields to the Report Header section
    - ShipName
    - ShipAddress
    - ShipCity
    - ShipRegion
    - ShipPostalCode
    - ShipCountry
    - CustomerName
    - Address
    - City
    - Region
    - PostalCode
    - Country
    - Salesperson
    - OrderDate
    - ShippedDate

- Add the following fields to the Details section
    - ProductName
    - UnitPrice
    - Quantity
    - Discount
    - ExtendedPrice



Arrange the fields on the report as shown below. Insert a subtotal field which sums the ExtendedPrice field. Place the subtotal field under in the Group Header section.



Next, create the form that will display the InvoiceDoc report in the browser. Add a new Web form to the project and name it “Document.aspx”. Add five controls to the form; two labels, two drop down lists, and one button. Set the following properties for the controls.

Button

Property Name	Value
Name	btnRunReport
Text	Run

Label

Property Name	Value
Name	lblCustomer
Text	Customer:

Label

Property Name	Value
Name	lblOrderID
Text	Order ID:

DropDownList

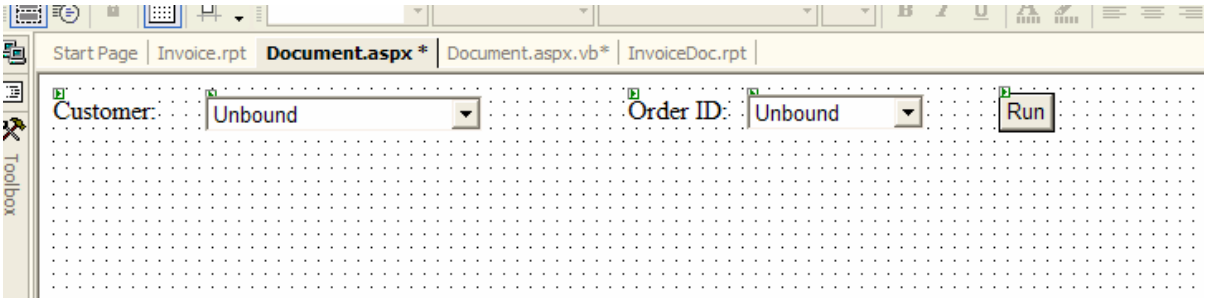
Property Name	Value
Name	cmbCustomer

AutoPostBack	True

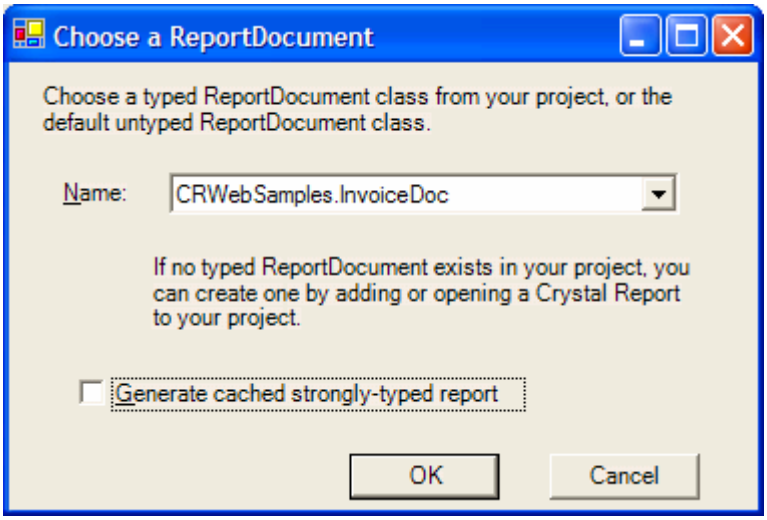
DropDownList

Property Name	Value
Name	cmbOrders

Arrange the controls as shown below.



Drag a ReportDocument object from the Toolbox and drop it on the Document.aspx form. Choose the **InvoiceDoc** typed ReportDocument object from the drop down and uncheck the **Generate cached strongly-typed report** option. Click **OK** to add the ReportDocument to the form.



Putting it Together

At this point, the report and the form are set up. The remaining steps involve populating the customer and order id controls with data from the database, binding the data returned from the stored procedure to the ReportDocument object, and exporting the report to PDF format. The following code shows the Document.aspx page load event. Code in this event populates the cmbCustomer control with customer names from the database.

```
    Private Sub Page_Load(ByVal sender As System.Object, _
```

```
    ByVal e As System.EventArgs) Handles MyBase.Load

      If Not Page.IsPostBack Then

          ' get customers from db and display in drop down list
          Dim daCustomers As New SqlClient.SqlDataAdapter( _
            "SELECT CustomerID, CompanyName FROM Customers", _
            DB_CONNECTION)

          Dim dsCustomers As New DataSet("Customers")

          daCustomers.Fill(dsCustomers)

          cmbCustomer.DataSource = dsCustomers
          cmbCustomer.DataTextField = "CompanyName"
          cmbCustomer.DataValueField = "CustomerID"
          cmbCustomer.DataBind()

          daCustomers.Dispose()
          dsCustomers.Dispose()

          ' Populate the orders drop down
          ' based on the first customer
          GetOrdersForCustomer()

      End If

  End Sub
```

After populating the cmbCustomer control, the code populates the cmbOrderID control based on the first customer returned from the database. The GetOrdersForCustomer method, shown below, populates the cmbOrderID control.

```
  Private Sub GetOrdersForCustomer()

      ' get orders from db and display in drop down list
```

```
        Dim daOrders As New SqlClient.SqlDataAdapter( _
          "SELECT OrderID FROM Orders WHERE customerid = @CustomerID", _
          DB_CONNECTION)
        Dim dsOrders As New DataSet("Orders")

        daOrders.SelectCommand.Parameters.Add( _
          "@CustomerID", cmbCustomer.SelectedValue)
        daOrders.Fill(dsOrders)

        cmbOrders.DataSource = dsOrders
        cmbOrders.DataTextField = "OrderID"
        cmbOrders.DataValueField = "OrderID"
        cmbOrders.DataBind()

        daOrders.Dispose()
        dsOrders.Dispose()

    End Sub
```

The GetOrdersForCustomer method populates the cmbOrderID dropdownlist with all of the order id's related to a specific customer. The customer parameter comes from the value selected in the cmbCustomer dropdownlist. In addition to being called from the Page_Load event, GetOrdersForCustomer is also called from the cmbCustomer's SelectedIndexChanged event, ensuring that whenever a new customer is selected from the list of customers, the related orders are retrieved and displayed in the cmbOrderID control. The code in the SelectedIndexChanged event is shown below.

```
    Private Sub cmbCustomer_SelectedIndexChanged( _
       ByVal sender As Object, _
       ByVal e As System.EventArgs) _
       Handles cmbCustomer.SelectedIndexChanged

        ' load the orders for a specific customer
        GetOrdersForCustomer()

    End Sub
```

Code behind the btnRunReport button gets the selected customer and order id from the cmbCustomer and cmbOrderID controls respectively. Those values are used as parameters for the GetInvoicesForCustomer_sp procedure. The results of executing the stored procedure are returned in the InvoiceDS typed dataset and then bound to the InvoiceDoc1 ReportDocument object. These steps are accomplished in the first half of the button's click event, shown in the following code.

```
Private Sub btnRunReport_Click( _
  ByVal sender As System.Object, _
  ByVal e As System.EventArgs) _
  Handles btnRunReport.Click

    ' get data from database into typed dataset
    Dim da As New SqlClient.SqlDataAdapter( _
      "GetInvoiceForCustomer_sp", DB_CONNECTION)
    Dim ds As New InvoiceDS

    ' set the stored procedure parameter values
    With da.SelectCommand
        .CommandType = CommandType.StoredProcedure
        .Parameters.Add("@CustomerID", cmbCustomer.SelectedValue)
        .Parameters.Add("@OrderID", cmbOrders.SelectedValue)
    End With

    ' map the generic "Table" to the typed
    ' dataset table named "GetInvoiceForCustomer_sp
    da.TableMappings.Add("Table", "GetInvoiceForCustomer_sp")

    ' fill the dataset
    da.Fill(ds)

    ' set the report document's data source to the dataset
    invoiceDoc1.SetDataSource(ds)
```

The remaining code in the click event formats the report as PDF and displays it in the browser. Here is where the report in this scenario differs from the reports described

above. The CrystalReportViewer object is not used to display the results. Instead, the report results are streamed out to the browser in PDF format.

The report is exported to a stream object using the ExportToStream method of the ReportDocument (invoiceDoc1) object. ExportToStream takes one parameter of the type, ExportFormatType. The ExportFormatType parameter sets the report's exported format type. In addition to PDF, exported formats include Excel, Word, HTML, and RichText.

```
st = invoiceDoc1.ExportToStream( _
     ExportFormatType.PortableDocFormat)
```

The stream returned from the ExportToStream method call is assigned to a local instance of a .NET Stream object (the st variable in the code) where it is converted into a byte array using the Stream's Read method. The byte array is written to the browser using the Response object's BinaryWrite method. As its name implies, the BinaryWrite method writes a string of binary characters to the HTTP output stream. In this case, the output stream is to the browser.

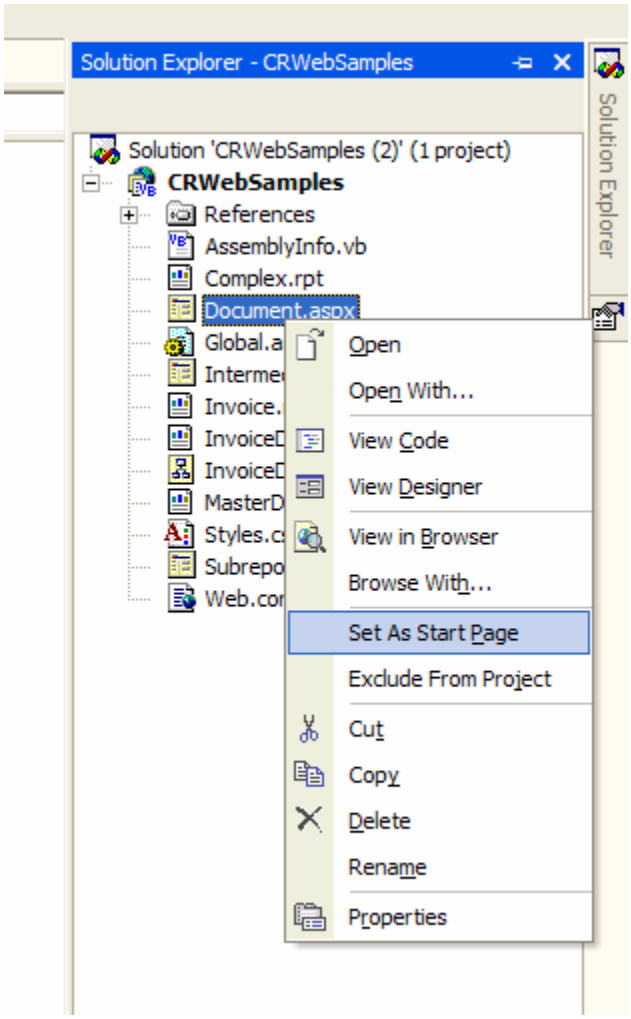
It is important to note that the ExportToStream method creates temporary files to export the report to the requested format. These files could hinder scalability in a Web application due to the overhead required to managing them.

To complete the solution, the MIME type is set to PDF using the Response object's ContentType property. This informs the browser that the stream data is in PDF format and starts the Acrobat Reader plug-in in Internet Explorer.

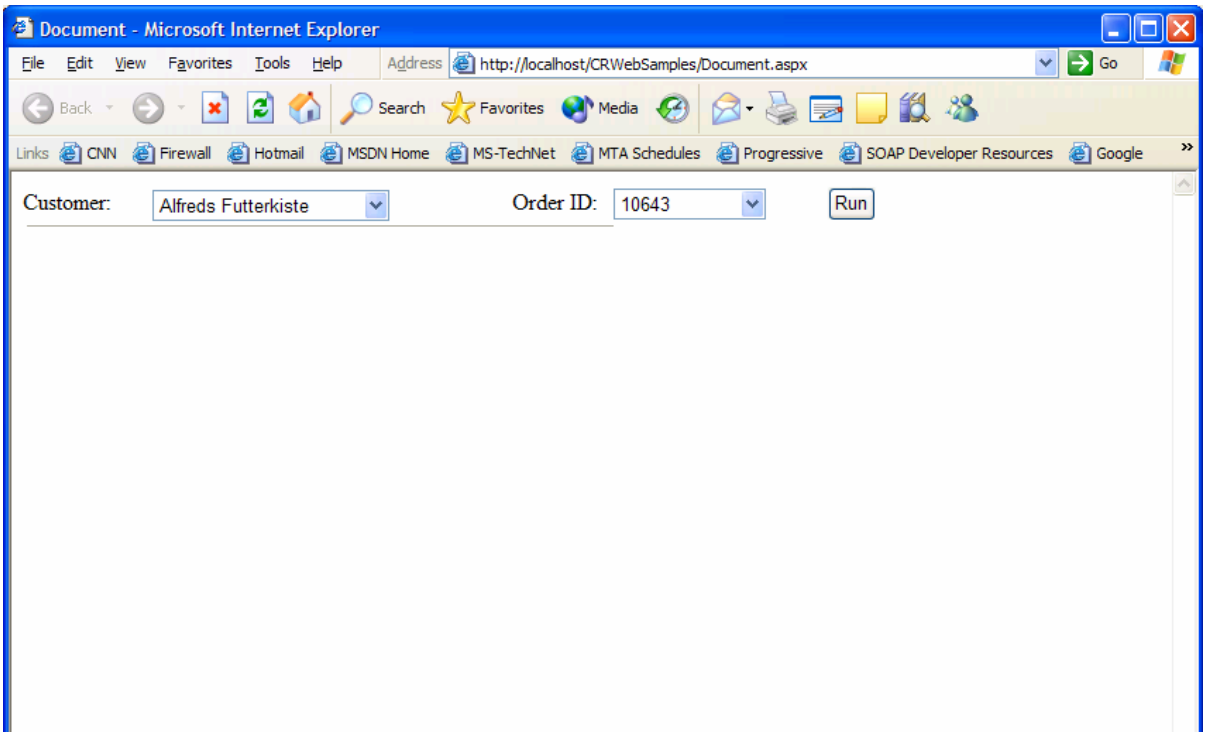
```
' create the stream object
Dim st As System.IO.Stream
st = invoiceDoc1.ExportToStream( _
  ExportFormatType.PortableDocFormat)

' set the http headers
Response.ClearContent()
Response.ClearHeaders()
' set the MIME type to PDF
Response.ContentType = "application/pdf"

' read the stream as binary (a byte array) and
' write the byte array to the browser
Dim b(st.Length) As Byte
st.Read(b, 0, st.Length)
```

```
        Response.BinaryWrite(b)
        Response.End()

    End Sub
```

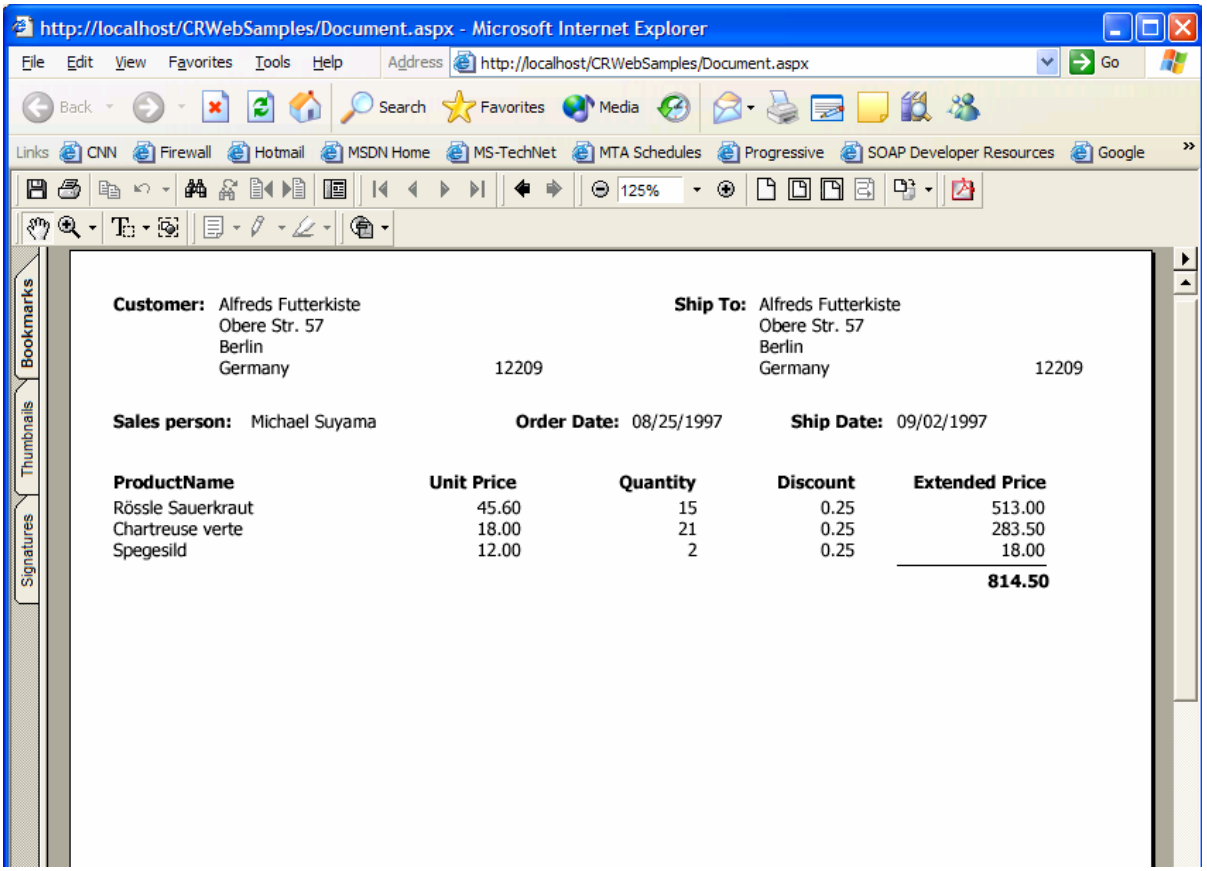
Before running the sample report, set the Document.aspx page as the start up page by right-clicking on the Document.aspx page in the Solution Explorer window and selecting the **Set as Start Page** as shown below.



Press the **F5** key to run the report. After the form loads (shown below), select a customer and an order id and click the **Run** button.



The report loads as a PDF document (shown below) instead of a Crystal Reports document.



The following list summarizes the steps taken in this section to create the sample report.

Paul Delcogliano, Progressive Systems Consulting, Inc., http://www.progsys.com

- Add a strongly typed DataSet to the project
    - Drop a stored procedure on the DataSet at design time from the Server Explorer.
- Add a new Crystal Report object to the project.
- Manually layout the report using the Blank Expert
    - Set the report's Data Source to the typed Dataset
    - Drag and drop fields from the Field Explorer to the report's details section
- Add a new web form to the project
    - Add controls to the form to run the report
    - Drag a ReportDocument component from the Toolbox and drop it on the web form. Set its typed ReportDocument class property to the proper typed report document object.
    - Add code to the form's Page_Load event to populate the dropdownlist controls.
    - Add code to the button's click event to retrieve data using the typed DataSet. Set the ReportDocument's ReportSource property to the populated DataSet
    - Add code to export the ReportDocument to a binary PDF stream. Set appropriate properties for the desired format. Stream the binary data to the browser using the Response object.

Scenario Conclusion

Generating legal documents is a powerful feature added to any application. The "Document" scenario was designed to walk through the steps required to produce legal quality documents using Crystal Reports. Crystal Reports' exporting features facilitate this process.

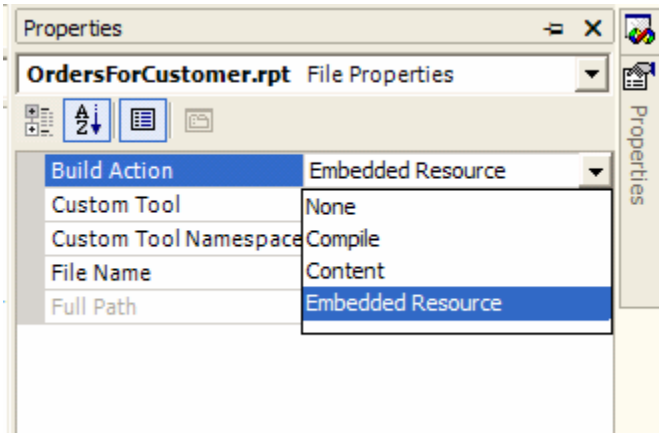
Furthermore, reports based on ADO.NET DataSets are a common requirement. This scenario reinforced the methods used for reporting on DataSets that were introduced in the "Sub-report" scenario above.

The reporting scenarios discussed above have all focused on developing robust reports using Crystal Reports for Visual Studio.NET 2003. Designing and developing reports is only half of a complete solution. Reports are worthless if they cannot be deployed. The next section discusses deployment issues found in different application environments and provides solutions to common deployment problems.

Deployment

By default, reports that are added to a .NET application are added as embedded resources. This means that the reports are compiled into the application's assembly. The reports will not be loaded from separate files. Embedding reports eliminates the need to distribute separate report files with the application. Applications take advantage of embedded reports through strongly typed ReportDocument objects. However, there is a disadvantage to embedding report in the assembly. Changes made to reports require the application to be recompiled and redeployed.

To prevent reports from being compiled into the assembly, select the report in the Solution Explorer and right-click on the report name. Choose the **Properties** option. From the Properties window, change the **Build Action** property from **Embedded Resource** to **None**.



Reports not compiled into the assembly are loaded using the ReportDocument's Load() method. Also, the report files must be manually added to the setup project to be distributed with the application. When reports are not compiled into the assembly, the application cannot use strongly typed ReportDocument objects, but can modify and redeploy a report without recompiling the application.

Merge Modules

Crystal Reports for Visual Studio.NET 2003 uses merge modules for deployment. Other deployment options are available with the stand alone version of Crystal Reports. Merge modules are provided for developers who want to include Crystal Reports runtime files along with their application.

A merge module is a simplified installer file. Merge modules install components used by the application. Crystal Reports merge modules need to be added to the setup project to successfully deploy an application that provides reporting functionality. Different versions of Crystal Reports require different merge module files. For Crystal Reports for Visual Studio.NET 2003, the following merge modules are required.

File Name	Description
Crystal_Managed2003.msm	Used to install the Crystal Reports for VS.NET managed components
Crystal_Database_Access2003.msm	Used to install database drivers used by the reports. Also installs export destinations

	and format drivers.
Crystal_Database_Access2003_enu.msm	Installs language specific components. Also installs PDF components.
Crystal_RegWiz2003.msm	Configures registration and licensing. A license key must be provided for this module when building a setup project.
VC_User_CRT71_RTL_X86_---.msm and VC_User_STL71_RTL_X86_---.msm	Optional. Required only when reports use ADO.NET DataSet objects.

By default, these files are located in the **C:\Program Files\Common Files\Merge Modules** folder.

License Key

When Crystal_RegWiz2003.msm is added to a setup project, it exposes a LicenseKey property. The LicenseKey property accepts a 19 digit license key value. The license key value was emailed when Crystal Reports was registered and can also be found in the **Help | About** menu in VS.NET 2003. Enter the license key value into the LicenseKey property.

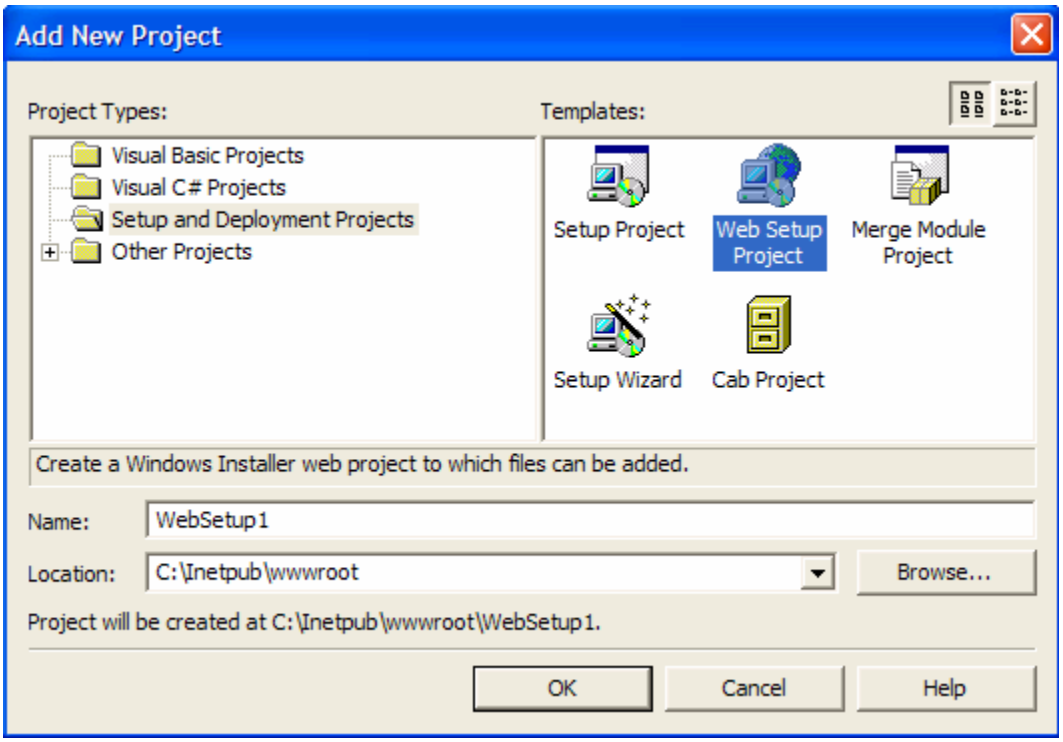
An error will occur when the setup project is built if the LicenseKey property is not set. The three most common errors are:

- "Invalid or missing KeycodeV2.dll" – see Kbase c2010681
- "Job Failed Because a Free License Could not be Obtained" – see Kbase c2012716
- "Err Msg: Cannot Find keycodev2.dll or invalid keycode" appears in VS.NET – see Kbase c2011205

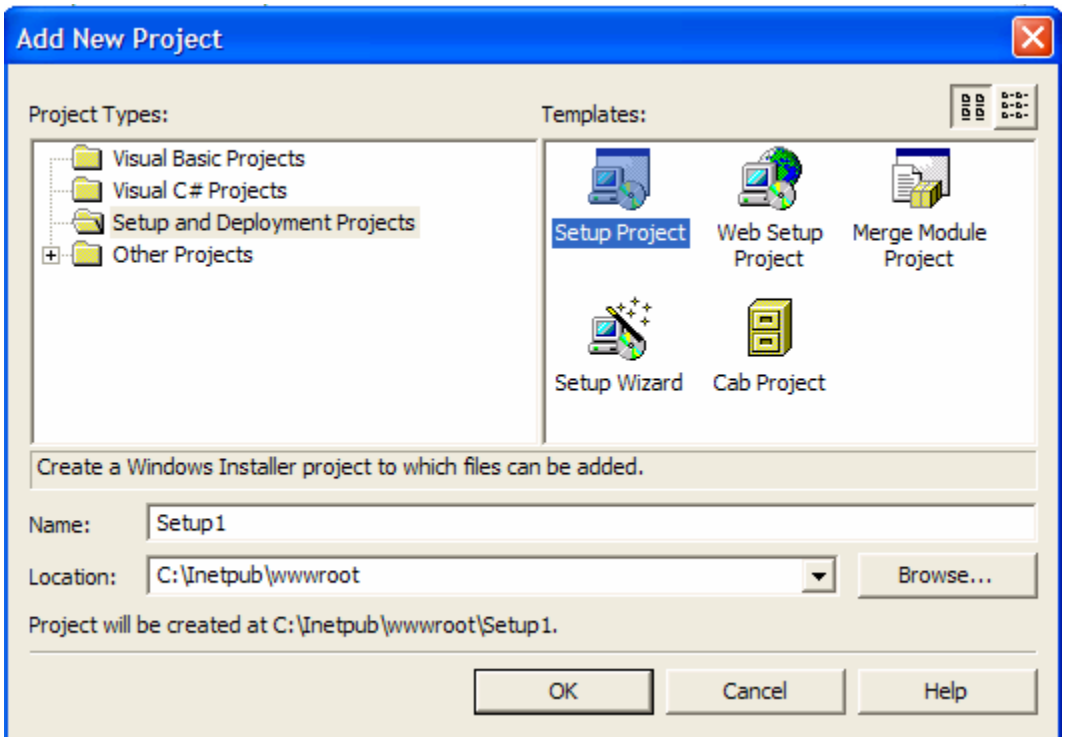
Application Deployment

This section discusses the steps needed to create a setup project to deploy the reporting application. The steps for deploying a Windows application are similar to those for deploying a Web application. Areas where the procedures differ are called out.

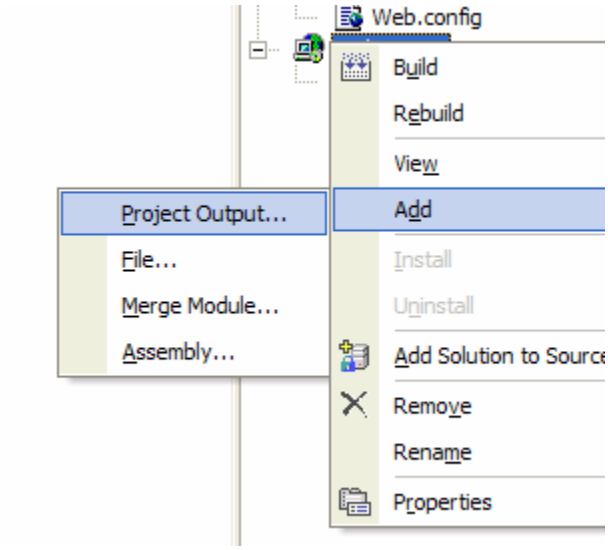
- Right click on the solution in the Solution Explorer window. Choose **Add | New Project...** from the menu.
- For a Web application, choose the **Setup and Deployment Projects** folder in the left pane, then select the Web Setup Project icon in the right pane.



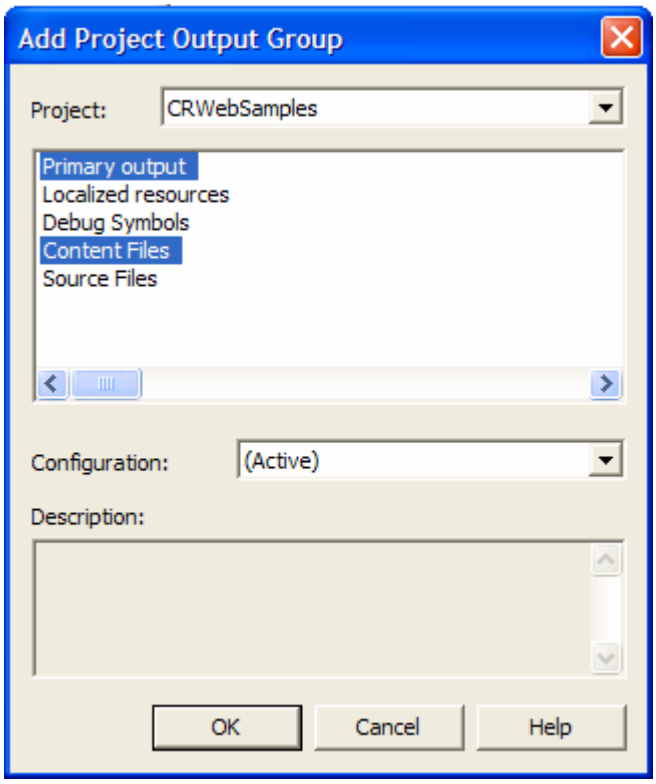
- For a Windows application, choose the **Setup and Deployment Projects** folder in the left pane, then select the Setup Project icon in the right pane.



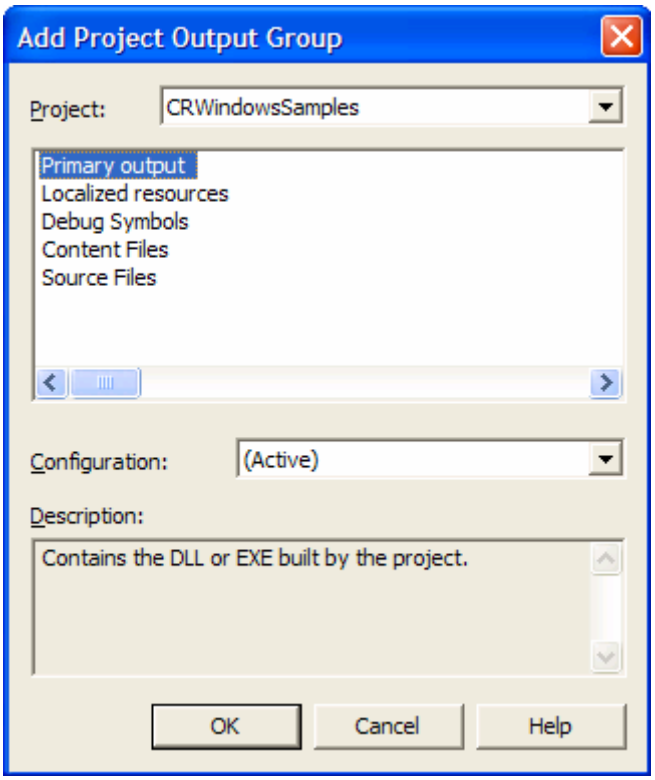
- Right-click on the Setup Project file in the Solution Explorer window and choose **Add > | Project Output...** from the menu.



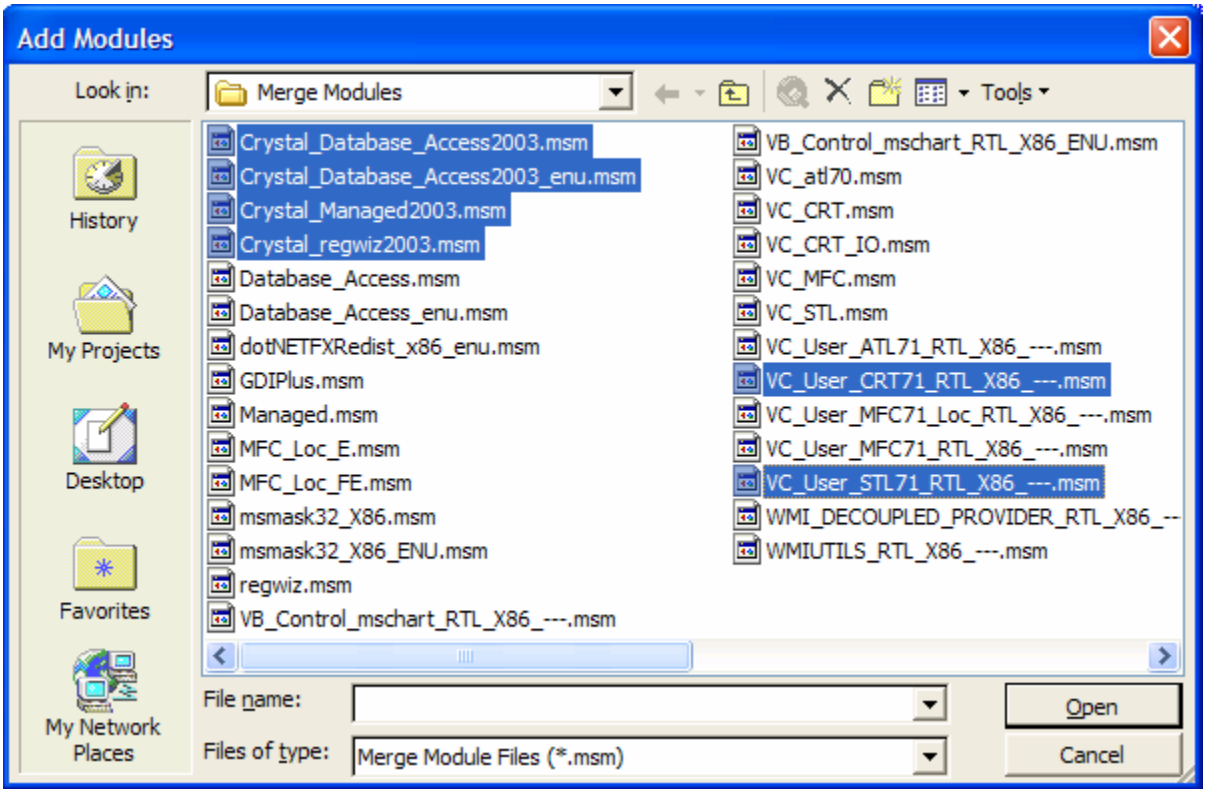
- For Web applications, select the **Primary output** and **Content Files** items from the list and click **OK**.



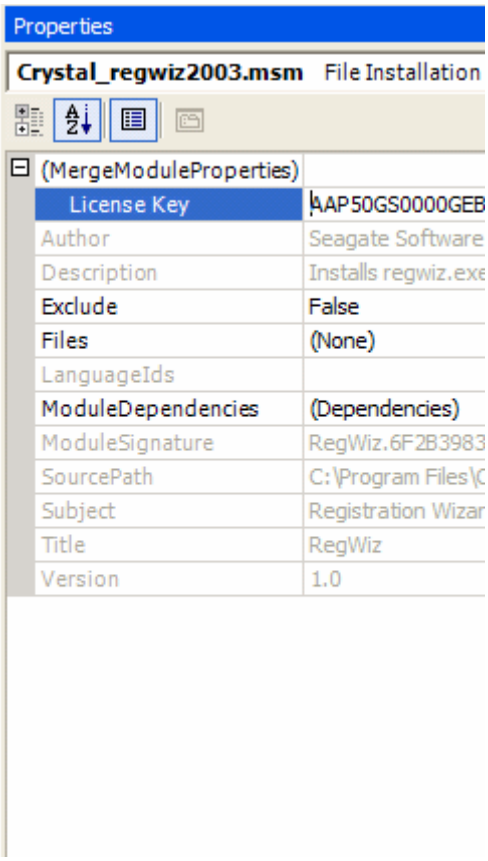
- For Windows applications, select the **Primary output** option from the list and click **OK**



- Project dependencies will be detected with the **OK** button is clicked.
- Add the merge modules listed above to the setup project by right-clicking the setup project in the Solutions Explorer window and selecting **Add > | Merge Module...** from the menu. Select the appropriate files from the dialog.



- If the application's reports use ADO.NET DataSets, add the VC_User_CRT71_RTL_X86_---.msm and VC_User_STL71_RTL_X86_---.msm merge modules files as well.
- Right click on the Crystal_RegWiz2003.msm file a view its properties. Expand the *MergeModuleProperties* property to enter the 19 digit license key value. The license key can be obtained from **Help | About** in VS.NET 2003.



To deploy the application, right click on the setup project in the Solution Explorer window and choose the **Build** option. Once the project is built, it is ready to deploy to the appropriate platform.

Conclusion

This document introduced Crystal Reports for Visual Studio.NET through five common reporting scenarios. The scenarios built upon one another. Each sample report described one or more features and discussed how to implement each feature in the report.

The Basic scenario created a simple Windows report listing customers from the sample database. Report Experts did most of the work designing the report. Data was retrieved directly from the customers table.

Drill down capabilities was the highlighted feature in the Intermediate scenario. Drill down enables users to view the detail records that make up a summary record. This report introduced on the idea of retrieving data from views and the ReportDocument object. The ReportDocument object exposes properties and methods to

programmatically run a report. This scenario also introduced Web based reporting using the CrystalReportViewer control.

The Visual scenario showed off some of Crystal Reports charting strengths with a chart of product sales for a specific product category. It introduced methods and code for creating reports based on a parameterized SQL Server stored procedure. The sample report expanded on the ReportDocument component that was introduced in the Intermediate scenario.

The Sub-report scenario demonstrated embedding a report within a report. Performance considerations were accounted for by configuring the sub-report to load on the click of a hyperlink.

The Document scenario highlighted Crystal's exporting features. Legal documents were created by exporting a report to PDF format and displaying the PDF data in a Web browser. Features of the ReportDocument object were used to stream the PDF output directly to the browser, eliminating the need to create temporary files to export the report.

Finally, the guide walked through the steps required to deploy applications with reporting capabilities. Reports embedded in the application assembly ease deployment but may hinder maintenance, while stand alone report files impede deployment but ease maintenance. In either case, merge modules are necessary to properly install the Crystal Reports dependency files during deployment.

This startup guide touches on many features available with Crystal Reports for Visual Studo.NET 2003. These features are designed to ease reporting development and deployment for programmers and enhance report presentation and functionality for end users.

About the Author

Paul Delcogliano is director of technology for Progressive Systems Consulting where he has been developing Windows and Web based applications with Visual Studio .NET since beta 1. Contact Paul at pdelco@progsys.com

For More Information

- Knowledgebase articles and additional sample code can be obtained at http://www.businessobjects.com/products/reporting/crystalreports/net/vsnet.asp
- More information about deployment is available at http://www.businessobjects.com/communityCS/TechnicalPapers/crnet_deployment.pdf
- Visit the DevZone for .NET developers who use products from Business Objects at http://www.businessobjects.com/products/dev_zone/net/default.asp

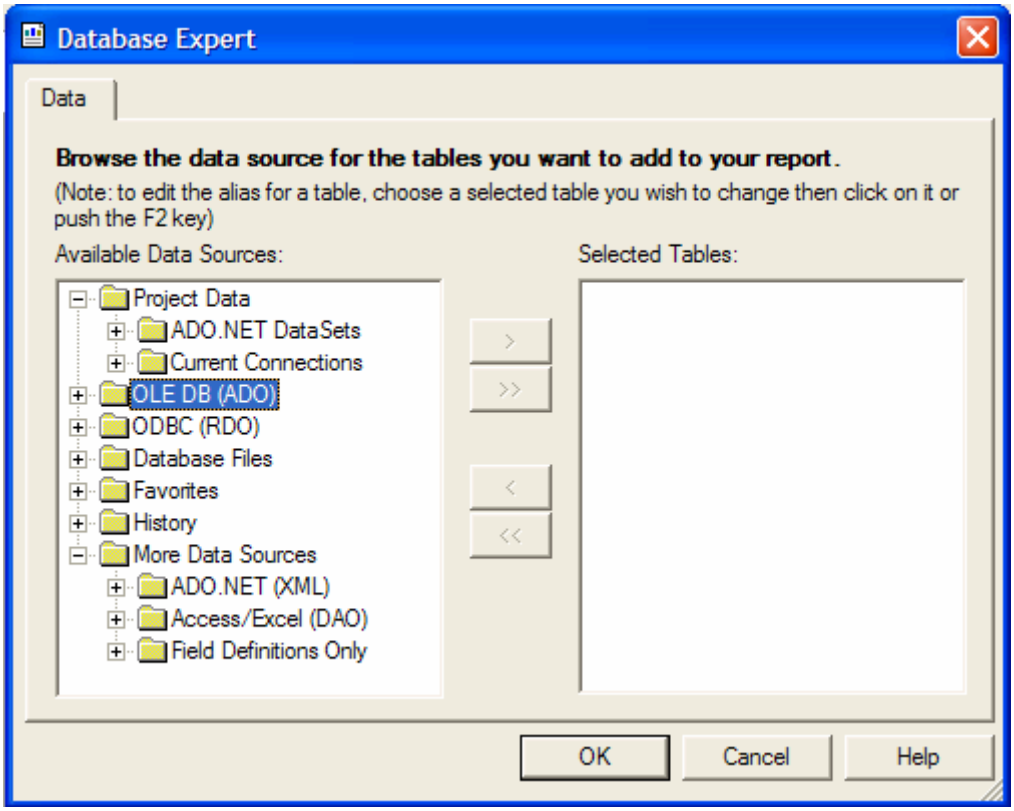
Appendix A

Database Connection Used by Windows Based Reports

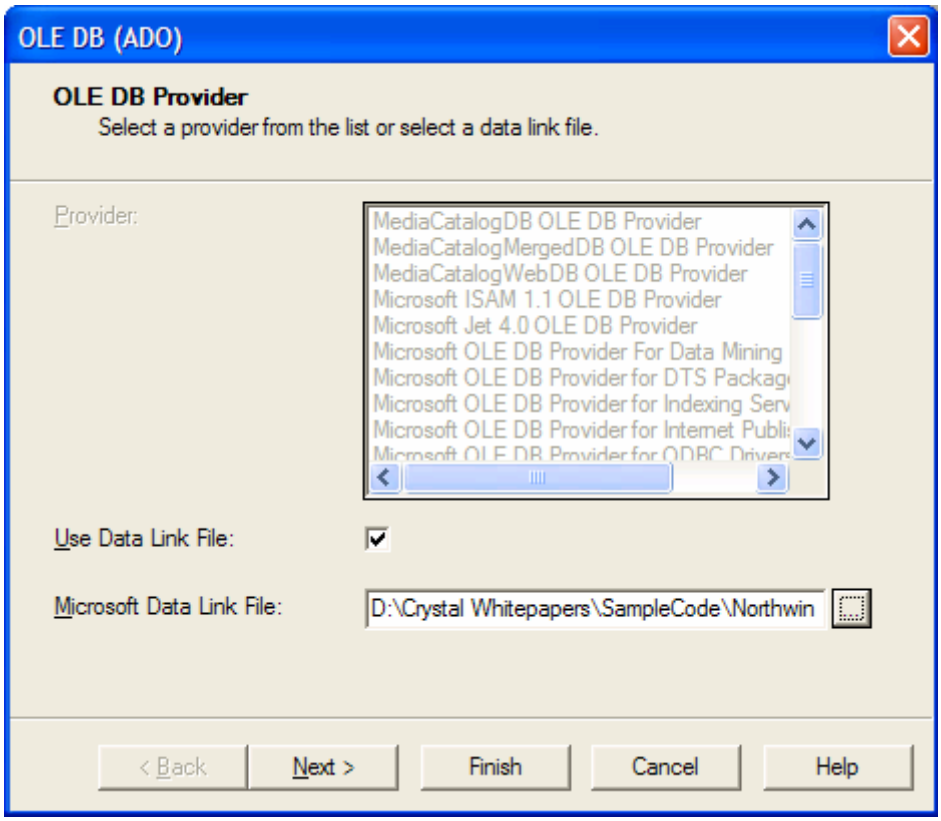
For simplicity, all Windows reports connect to the database using a data link file. The file, “Northwind.udl”, accompanies the sample code. The file connects to the Northwind database on a local workstation. It uses Windows authentication to connect to the database. The information in the data link file may need to be adjusted depending on the development environment.

For sample Windows applications, connecting to the database is done using the data link file. Connecting the report to the database occurs during report creation, either with the help of a report Expert or manually. In either case, the following steps will connect the report to the database using the data link file.

Click on the OLE DB (ADO) item in the list of available data sources.



A new form will appear, prompting for the selection of an OLE DB provider. Instead of selecting a provider, check the **Use Data Link File** checkbox. Browse to the Northwind.udl file included with the sample code and click **Finish**.



After selecting the connection information, click the **Finish** button.

Now that the connection is established, begin selecting the report's data source as described in the scenarios above.

Database Connection Used by Web Based Reports

The connection used by the Web based report samples is similar to the one used by the Windows report samples. The biggest difference is in the method used to authenticate the user. The Windows sample reports use Windows authentication to connect to SQL Server. Windows authentication is not a scalable solution for a Web application. Scalability is limited because each user that accesses the Web site needs a Windows account. This may be okay for an Intranet application where the number of users is small. But in an Internet application, Windows authentication doesn't work well. Rather than using Windows authentication to connect to SQL Server, the sample Web reports use SQL Server authentication.

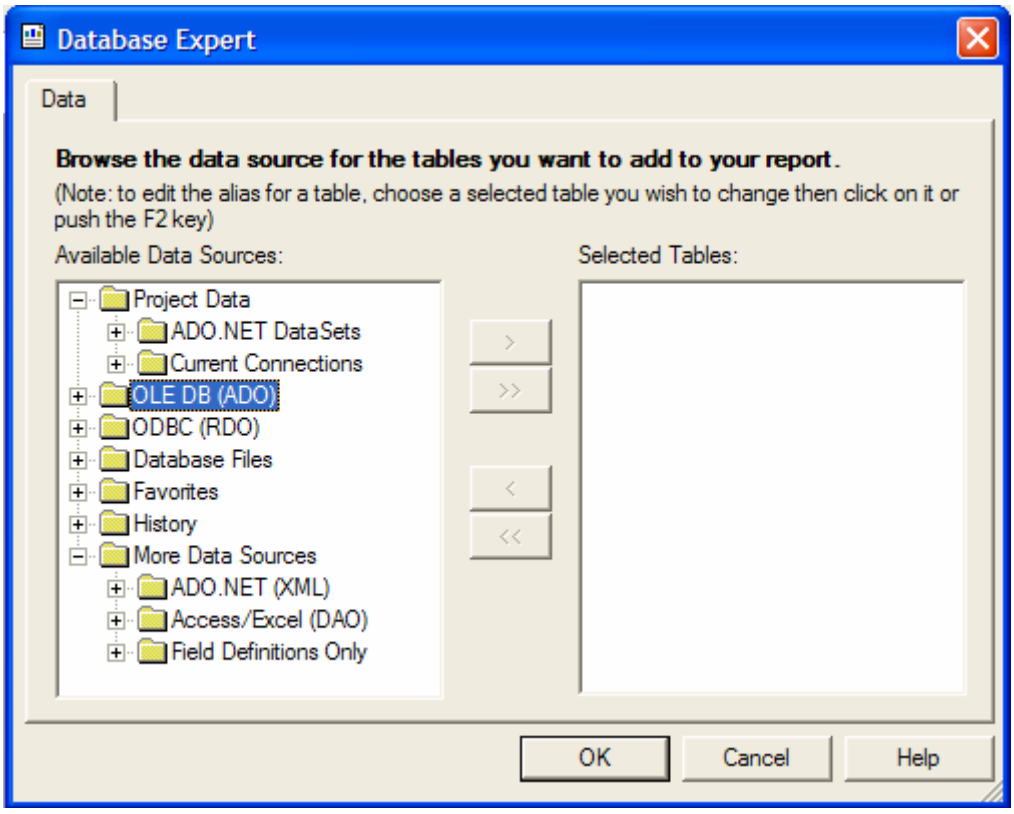
With SQL Server authentication, SQL Server performs the authentication itself by checking to see if a SQL Server login account has been set up and if the specified password matches the password on file. If so, authentication is successful.

Authentication can be performed in a variety of methods depending on the server environment. An article describing best practices for accessing SQL Server from ASP.NET applications can be found on MSDN.

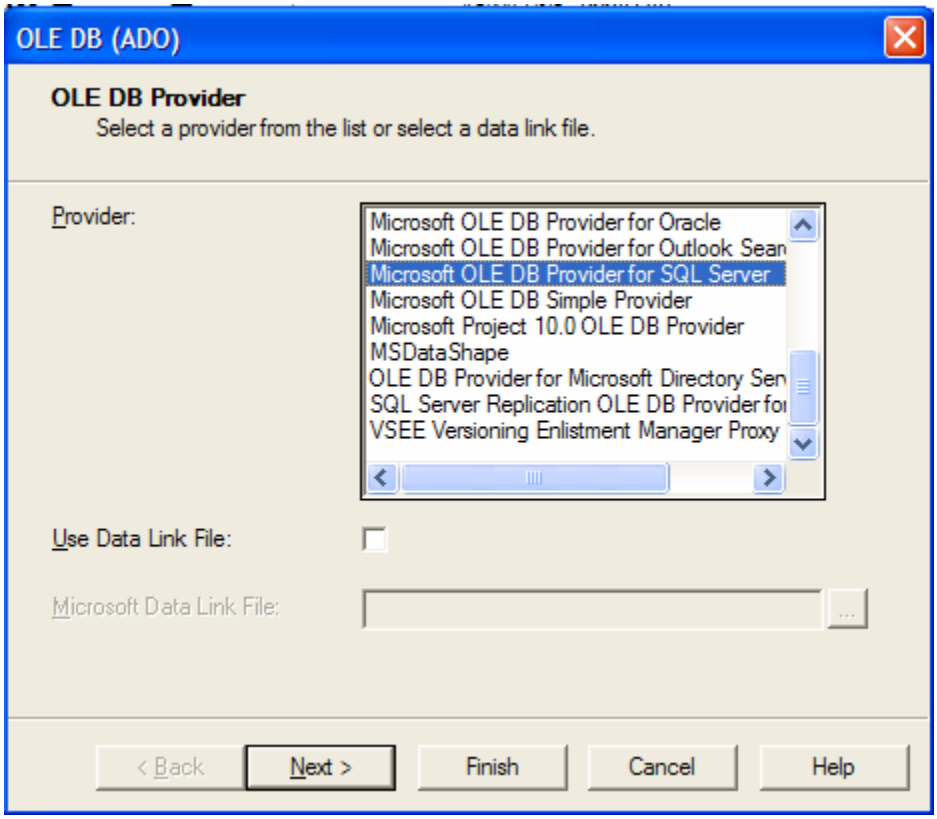
The samples authenticate with a SQL Server login named "CRReports". This account only has access to the Northwind database. A common practice is to use the system administrator account, a.k.a. "sa". For security reasons, this is not a good practice.

For sample Web applications, connecting to the database is done using the Database Expert. Connecting the report to the database occurs during report creation, either with the help of a report Expert or manually.

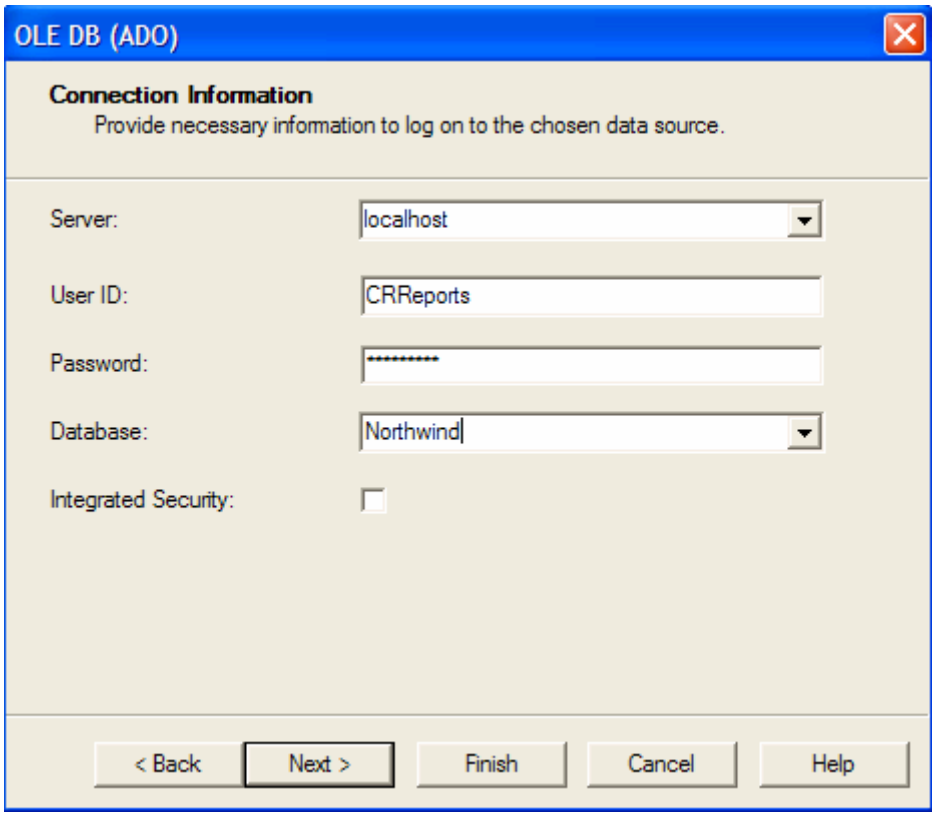
Click on the OLE DB (ADO) item in the list of available data sources.



Expand the *OLE DB (ADO)* node. Select the **OLE DB Provider for SQL Server** from the list of providers.



Click the **Next >** button to enter the connection information.



After entering the connection information, click the **Finish** button.

Now that the connection is established, begin selecting the report's data source as described in the scenarios above.