Crystal Reports

For Visual Studio 2005

Walkthroughs

Version 1.0

Contents

Crystal Reports

For Visual Studio 2005

CrystalReportViewer Object Model Tutorials

Crystal Reports

For Visual Studio 2005

CrystalReportViewer Object Model Tutorial:

Logging onto a Secure SQL Server Database

Logging onto a Secure SQL Server Database

Introduction

When you use the CrystalReportViewer object model to log on to a secure SQL Server database, you must use classes from the CrystalDecisions.Shared namespace.

The properties of the ConnectionInfo class are used to connect to the database server or ODBC data sources. In this tutorial, you set the ServerName, DatabaseName, UserID and Password properties. If you choose only to set the DatabaseName, UserID, and Password properties, then you are logged on to the default server of the database that you have specified within the report. However, if you choose to assign an alternate ServerName property, you can redirect the report to a different server at runtime.

You retrieve the TableLogOnInfos instance from the LogOnInfo property of the CrystalReportViewer control. Then, you assign the ConnectionInfo instance to each TableLogOnInfo object in the TableLogOnInfos instance.

The logon code in this tutorial also works for subreports that are inserted into the main report. For the ReportDocument object model, you must modify the code to work for subreports. The ReportDocument object model is recommended over the CrystalReportViewer object model.

Creating a Report Connected to a Secure SQL Server Database

To begin, create a report that draws its information from the Northwind database.

Note Northwind is a sample database provided with SQL Server.

Some setup is required as a prerequisite to this tutorial.

Prerequisite Database Setup

1. SQL Server configuration:

   If you have SQL Server (or the OEM version, MSDE) installed, it must be configured to require SQL Server Authentication.

   If you do not have SQL Server (or the OEM version, MSDE) installed, you must install MSDE with SQL Server Authentication set to "True."
2. The Northwind database provided with SQL Server must be installed and verified that it requires SQL Server Authentication.
3. A limited access account must be created for use within the Web site.

To install MSDE with SQL Server Authentication, or the Northwind database, go to the following sections from Appendix: System Setup in this documentation:

Appendix: MSDE Installation with Windows or SQL Server Authentication

Appendix: Northwind Database Installation

Appendix: Security: Creating a Limited Access Database Account

Once you have configured SQL Server and the Northwind database according to the sections above, you are ready to create a report that draws its information securely from a Northwind database.

To create a report with secure data from the Northwind database

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the **Crystal Report** template.
3. In the **Name** field, enter the name "NorthwindCustomers.rpt", and then click **Open**.

   Note If you have not registered before, you may be asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
4. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
5. In the **Choose an Expert** panel, select **Standard**. Click **OK**.

   The Standard Report Creation Wizard window appears.
6. In the **Available Data Sources** panel, expand the **Create New Connection** folder.
7. From the subfolder that opens, expand the **OLE DB (ADO)** folder.

   The OLE DB (ADO) window appears.
8. Select **Microsoft OLE DB Provider for SQL Server**, and then click **Next**.
9. Enter the values for your database server, user id and password into the **Server**, **User ID** and **Password** fields.
10. From the **Database** drop down list, select "Northwind."

    Leave the Integrated Security checkbox unchecked because you are using SQL Server authentication instead of NT authentication.
11. Click **Finish**.

    The OLE DB folder is now expanded, showing your database server, and within it, the Northwind database.
12. Expand the nodes **Northwind**, **dbo**, and **Tables**, and then select the **Customers** table.
13. Click the **>** symbol to move the table into the **Selected Tables** panel, and then click **Next**.
14. Expand the **Customers** table, and then hold down the Ctrl key while you click **CompanyName**, **ContactName** and **City**.
15. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click **Next**.
16. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, then click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

    The NorthwindCustomers report is created and loaded into the main window of Visual Studio.

You are now ready to bind the report to the CrystalReportViewer control.

Binding the Report Without Logon Code

In Appendix: Project Setup, you have placed a CrystalReportViewer control on the Web or Windows Form. In the previous step, you have added a NorthwindCustomers report to the project.

In this section, you bind the file directory path of the NorthwindCustomers report to the CrystalReportViewer control. Then you test whether the report displays correctly when the database logon code has not been set.

To bind the file directory path of the NorthwindCustomers report to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Locate the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup).
4. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("NorthwindCustomers.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("NorthwindCustomers.rpt");
```

[end]

   For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

   Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"NorthwindCustomers.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"NorthwindCustomers.rpt";
```

[end]

5. Assign the file directory path of the NorthwindCustomers report to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = reportPath
```

[end]

[C#]

```
crystalReportViewer.ReportSource = reportPath;
```

[end]

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to log onto the database.

To test the loading of the NorthwindCustomers report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\debug\` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The NorthwindCustomers report will not display. It will display after you add the database logon code.

   Note Results may vary, depending on the version of Crystal Reports that you use. For example, if you have Crystal Reports 10 and up installed, you are prompted for database logon information for that report. This is a new feature of Crystal Reports Developer and Crystal Reports for Visual Studio 2005. If you are running a previous version of Crystal Reports, an exception is thrown. In either case, you must follow the next step procedure to create a fully functional application.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Adding the Report Logon Code

You are now ready to add the logon code to the code-behind class. You begin by creating a private helper method, `SetDBLogonForReport()`.

To create and code the SetDBLogonForReport() method

1. Return to the code-behind class for this Web or Windows Form.
2. At the bottom of the class, create a new private method named `SetDBLogonForReport()` with ConnectionInfo passed into the method signature.

[Visual Basic]

```
Private Sub SetDBLogonForReport(ByVal myConnectionInfo As ConnectionInfo)


End Sub
```

[end]

[C#]

```
private void SetDBLogonForReport(ConnectionInfo connectionInfo)
```

```
    {

    }
```
[end]

3. Within this method, retrieve the TableLogOnInfos instance from the LogOnInfo property of the CrystalReportViewer class.

   Note To make the TableLogOnInfos class accessible, include an `"Imports"` `[Visual Basic]` or `"using"` `[C#]` statement at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

   TableLogOnInfos is an indexed class that contains instances of the TableLogOnInfo class.

[Visual Basic]
```
    Dim myTableLogOnInfos As TableLogOnInfos =
    myCrystalReportViewer.LogOnInfo
```
[end]
[C#]
```
    TableLogOnInfos tableLogOnInfos = crystalReportViewer.LogOnInfo;
```
[end]

4. Create a `foreach` loop that loops through each TableLogOnInfo instance in the TableLogOnInfos indexed class instance.

[Visual Basic]
```
    For Each myTableLogOnInfo As TableLogOnInfo In myTableLogOnInfos


    Next
```
[end]
[C#]
```
    foreach(TableLogOnInfo tableLogOnInfo in tableLogOnInfos)
    {

    }
```
[end]

5. Within the `foreach` loop, set the ConnectionInfo property of TableLogOnInfo to the ConnectionInfo parameter.

[Visual Basic]
```
    myTableLogOnInfo.ConnectionInfo = myConnectionInfo
```
[end]
[C#]
```
    tableLogOnInfo.ConnectionInfo = connectionInfo;
```
[end]

This step procedure has created a method to set the logon for the database. However, you must modify the `ConfigureCrystalReports()` method to address this method, for the report to be aware that it has database logon information.

Modifying the ConfigureCrystalReports() method requires two actions:

Configure the ConnectionInfo instance.

Call the SetDBLogonForReport() method.

To modify the ConfigureCrystalReports() method to address the database logon code

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
2. Within the line breaks, declare and instantiate the ConnectionInfo class.

[Visual Basic]

```
Dim myConnectionInfo As ConnectionInfo = New ConnectionInfo()
```

[end]

[C#]

```
ConnectionInfo connectionInfo = new ConnectionInfo();
```

[end]

3. Set the DatabaseName, UserID, and Password properties of the ConnectionInfo instance.

   Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

   In the code that you write, replace the sample *1234* password (shown below) with your own password.

[Visual Basic]

```
myConnectionInfo.DatabaseName = "Northwind"
myConnectionInfo.UserID = "limitedPermissionAccount"
myConnectionInfo.Password = "1234"
```

[end]

[C#]

```
connectionInfo.DatabaseName = "Northwind";
connectionInfo.UserID = "limitedPermissionAccount";
connectionInfo.Password = "1234";
```

[end]

4. Below the code that binds the CrystalReportViewer control, call the `SetDBLogonForReport()` method, by passing in the ConnectionInfo instance.

[Visual Basic]

```
SetDBLogonForReport(myConnectionInfo)
```

[end]

[C#]

```
SetDBLogonForReport(connectionInfo);
```

[end]

You are now ready to build and run your project. The report should load properly, because you have added code to log on to the database.

To test the loading of the NorthwindCustomers report

1. From the Build menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you learn how to change the database location at runtime.

Adding Ability to Change Database Location at Runtime

In this section, you learn how to change the database location at runtime. This requires only a minor modification to the ConnectionInfo instance.

To change the database location at runtime

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the line that declares and instantiates the ConnectionInfo class.
2. Within the line breaks, set the ServerName property of the ConnectionInfo instance.

   Note In the code that you write, replace the sample server name *DevDatabase* (shown below) with the name of your server.

[Visual Basic]

```
myConnectionInfo.ServerName = "DevDatabase"
```

[end]

[C#]

```
connectionInfo.ServerName = "DevDatabase";
```

[end]

You are now ready to build and run your project. The report should redirect to the alternate database server at runtime.

To test that the report can be reset to an alternate database server at runtime

1. From the Build menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully set your code to log onto the SQL Server database and change database location at runtime.

When you can add subreports to your main report (NorthwindCustomer.rpt), you do not need additional code.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_CRVObjMod_DBLogon

C# Windows project: CS_Win_CRVObjMod_DBLogon

Visual Basic Web Site: VB_Web_CRVObjMod_DBLogon

Visual Basic Windows project: VB_Win_CRVObjMod_DBLogon

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

CrystalReportViewer Object Model Tutorial:

Reading and Setting Discrete Parameters

Reading and Setting Discrete Parameters

Introduction

A discrete parameter is a single value, as opposed to a range parameter, which refers to a range of values. Text items (such as cities) are usually accessed through discrete parameters. Numeric items (such as employee salaries) are usually accessed through range parameters.

In this tutorial, you set a value for a discrete parameter to view a customer report that is based on one field. The report displays only those customers who live in the cities you have selected from a city list. The city list is derived from the default values for the City parameter. Those default values for the City parameter are encapsulated within the report.

To begin, you create a customer report with a city parameter. The report derives its data from the sample database that is shipped with Crystal Reports for Visual Studio. When you instantiate the report in the code, you create an ArrayList that contains city names (Paris, Tokyo) and pass that ArrayList instance to a helper method that sets those city names as the current values for the city parameter. You then bind the report to the CrystalReportViewer control and view the report with only those customers that live in Paris and Tokyo displayed.

In the next part of the tutorial, you do the following:

Create a method that gets all the default values and returns them in an ArrayList.

Add a ListBox control to the form and populate it from the ArrayList.

Add a Button control to redisplay the report based on ListBox selections.

In the final part of the tutorial, you code the button click event to retrieve any selected items from the ListBox control and set those to be the current values for the city parameter. The report redisplays and shows only those customers who live in the cities that have been selected within the ListBox control.

This tutorial can also be completed with classes of the ReportDocument object model.

Creating a Report with Parameters

To begin, create a report that draws its information from the sample Microsoft Access database that ships with Crystal Reports.

Note Xtreme.mdb is the sample database that is provided with Crystal Reports. To locate xtreme.mdb on your hard drive for your version of Crystal Reports, see Appendix: Location of Xtreme Sample Database. You need to connect to the database through its ODBC DSN entry. To learn the name of this entry for your version of Crystal Reports, see Appendix: ODBC DSN Entry for Xtreme Sample Database.

To create a report with parameters

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the **Crystal Report** template.
3. In the **Name** field, enter the name "CustomersByCity.rpt" and click **Add**.

   Note In Visual Studio .NET 2002 or 2003, the button is named Open.

   If you have not registered before, you are asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
4. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
5. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
6. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to Crystal Reports Developer, the Create New Connection folder does not exist; the contents are shown at the top level.
7. From the subfolder that opens, expand the **ODBC (RDO)** folder.
8. In the ODBC (RDO) window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
9. Expand the **Tables** node, double-click the **Customer** table to move the table to the **Selected Tables** panel, and then click **Next**.
10. Expand the **Customer** table, then CTRL-click **Customer Name**, **Contact Title**, **Address1**, **Contact Last Name** and **City**.
11. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click the **Next** button.
12. In the **Available Fields** panel, under **Report Fields**, double-click **Customer.City** to move the field into the **Group By** panel, and then click the **Finish** button.

    The CustomersByCity report is created and loaded into the main window of Visual Studio.

You are now ready to add a parameter named City and populate it with default values.

The Field Explorer must be visible, because it provides access to the various features of the report, including parameters. To make the Field Explorer visible, from the **Crystal Reports** menu, click **Field Explorer**.

To add a City parameter

1. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click **Toggle Field View**.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.
2. In the **Field Explorer**, right-click **Parameter Fields** and select **New...**
3. In the **Create Parameter Field** dialog box:

   Set the **Name** to "City."

Set the **Prompting Text** to "Select one or more cities."

Set **Value Type** to **String**

Select **Allow multiple values**.

Select **Discrete value(s)**.

Click **Default Values...**

Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to Crystal Reports Developer, this button is named Set Default Values.

4. In the **Set Default Values** dialog box:

   Set the **Browse table** to "Customer."

   Set the **Browse field** to "City."

   Click **>>** (the double right arrow) to move the entire cities list into the **Default Values** list.

5. Click **OK** to close the **Set Default Values** dialog box.
6. Click **OK** to close the **Create Parameter Field** dialog box.

You have just set the Default Values to contain a large list of cities. Later in this tutorial, you access this same list of default values programmatically, through the ParameterFieldInfo property of the CrystalReportViewer class.

You now use the Select Expert to set a formula that connects the City database column to your newly created City parameter field.

To connect the City parameter to the City database column

1. On the Crystal Reports toolbar, click **Select Expert**.
2. In the **Choose Field** dialog box, under **Report Fields**, select **Customer.City**, and then click **OK**.
3. In the **Select Expert** dialog box, within the **Customer.City** tab, set the dropdown list to "is equal to."
4. In the new dropdown list that appears to the right, select the first choice in the list, **{?City}**, and then click **OK**.

   Note This selection, {?City}, is the City parameter that you created earlier.

5. From the **File** menu, select **Save All**.

You are now ready to bind the report to the CrystalReportViewer control and set the city parameter with two initial values, Paris and Tokyo.

Binding the Report

When you followed the instructions in Appendix: Project Setup to prepare for this tutorial, you have placed a CrystalReportViewer control on the Web or Windows Form. In the previous steps, you added a CustomersByCity report to the project.

In this section, you bind the file directory path of the CustomersByCity report to the CrystalReportViewer control. Then you test whether the report displays correctly when current values have not been set for its parameter field.

To bind the file directory path of the CustomersByCity report to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code** to view the code-behind class for this Web or Windows Form.
3. Locate the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup).
4. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("CustomersByCity.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("CustomersByCity.rpt");
```

[end]

   For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

   Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"CustomersByCity.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"CustomersByCity.rpt";
```

[end]

5. Assign the file directory path of the CustomersByCity report to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = reportPath
```

[end]

[C#]

```
crystalReportViewer.ReportSource = reportPath;
```

[end]

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to set a value for the City parameter field.

To test the loading of the CustomersByCity report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The CustomersByCity report does not display. It displays after you add a value for the City parameter field later in this tutorial.

   Note Results may vary, depending on the version of Crystal Reports that you use. In more recent versions, you can see a form requesting that you provide parameter values for that report. In earlier versions, a "Missing parameter field current value" exception is thrown. In either case, you must add further code to create a fully functional application.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Setting Parameters Manually in Code

You are now ready to set two values ("Paris" and "Tokyo") into the City parameter field for the CustomersByCity report.

This involves some coding, which you can separate into the following processes:

You need a PARAMETER_FIELD_NAME constant to hold the "City" parameter field name.

The code to add current values to the City parameter is used at different locations in this tutorial; therefore, you create this code as a separate helper method.

Within the `ConfigureCrystalReports()` method, you add the "Paris" and "Tokyo" parameters to an ArrayList instance and pass in both the report and the ArrayList instance to the helper method to be processed.

To create a PARAMETER_FIELD_NAME constant

1. Return to the code-behind class for this Web or Windows Form.
2. At the class level, create a new string constant, PARAMETER_FIELD_NAME, and set its value to "City."

[Visual Basic]

```
Private Const PARAMETER_FIELD_NAME As String = "City"
```

[end]

[C#]

```
private const string PARAMETER_FIELD_NAME = "City";
```

[end]

You are now ready to create the helper method that adds current values to the parameter in the report.

To create a helper method that adds current values to the parameter in the report

1. Return to the code-behind class for this Web or Windows **Form**.
2. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class for the System.Collections namespace (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Collections
```

[end]

[C#]

```
using System.Collections;
```

[end]

Note This declaration is needed, to access the ArrayList class.

3. At the bottom of the class, create a new private method named `SetCurrentValuesForParameterField()`, with two variables in the method signature: ParameterFields and ArrayList.

[Visual Basic]

```
Private Sub SetCurrentValuesForParameterField(ByVal myParameterFields As
ParameterFields, ByVal myArrayList As ArrayList)


End Sub
```

[end]

[C#]

```
private void SetCurrentValuesForParameterField(ParameterFields
parameterFields, ArrayList arrayList)
{

}
```

[end]

4. Within this method, declare and instantiate the ParameterValues that are indexed class as the variable currentParameterValues.

Note For the ParameterValues class to be accessible, you must have included an `"Imports" [Visual Basic]` or `"using" [C#]` declaration at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

[Visual Basic]

```
Dim currentParameterValues As ParameterValues = New ParameterValues()
```

[end]

[C#]

```
ParameterValues currentParameterValues = new ParameterValues();
```

[end]

5. Create a `foreach` loop to retrieve all of the submitted values (as type Object) from the ArrayList instance.

   Note In this method, you retrieve values from the ArrayList. Later you write code that adds values to the ArrayList.

[Visual Basic]

```
For Each submittedValue As Object In myArrayList


Next
```

[end]

[C#]

```
foreach(object submittedValue in arrayList)
{
}
```

[end]

6. Within the `foreach` loop, declare and instantiate the `ParameterDiscreteValue` class.

[Visual Basic]

```
Dim myParameterDiscreteValue As ParameterDiscreteValue = New
ParameterDiscreteValue()
```

[end]

[C#]

```
ParameterDiscreteValue parameterDiscreteValue = new
ParameterDiscreteValue();
```

[end]

7. Within the `foreach` loop, convert the `submittedValue` to string and pass it to the **Value** property of the `ParameterDiscreteValue` instance.

[Visual Basic]

```
myParameterDiscreteValue.Value = submittedValue.ToString()
```

[end]

[C#]

```
parameterDiscreteValue.Value = submittedValue.ToString();
```

[end]

8. Within the `foreach` loop, add the ParameterDiscreteValue instance into the currentParameterValues indexed class.

[Visual Basic]

```
currentParameterValues.Add(myParameterDiscreteValue)
```

[end]

[C#]

```
currentParameterValues.Add(parameterDiscreteValue);
```

[end]

   This completes the code within the `foreach` loop. You place the remaining code (from the steps that follow) after the `foreach` loop.

9. Outside the `foreach` loop, retrieve the `ParameterField` instance from the `ParameterFields` indexed class that is based on the index entry of the PARAMETER_FIELD_NAME constant.

[Visual Basic]

```
Dim myParameterField As ParameterField =
myParameterFields(PARAMETER_FIELD_NAME)
```

[end]

[C#]

```
ParameterField parameterField = parameterFields[PARAMETER_FIELD_NAME];
```

[end]

10. Pass the `currentParameterValues` instance to the `CurrentValues` property of the `ParameterField` instance.

[Visual Basic]

```
myParameterField.CurrentValues = currentParameterValues
```

[end]

[C#]

```
parameterField.CurrentValues = currentParameterValues;
```

[end]

This step procedure showed you how to create a method that retrieves submitted values from an ArrayList instance and places them as current values into a ParameterField instance. Now, you must call this method before your report is bound to the CrystalReportViewer control, for the report to be aware that it has parameter settings.

To call the SetCurrentValuesForParameterField() method before the report is bound to the CrystalReportViewer control

1. In the `ConfigureCrystalReports()` method, declare and instantiate an ArrayList above the line that binds the report to the CrystalReportViewer control.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

2. Add the city names "Paris" and "Tokyo" as strings to the ArrayList instance.

[Visual Basic]

```
myArrayList.Add("Paris")
myArrayList.Add("Tokyo")
```

[end]

[C#]

```
arrayList.Add("Paris");
arrayList.Add("Tokyo");
```

[end]

3. Below the code that binds the **CrystalReportViewer** control, retrieve the `ParameterFields` instance from the `ParameterFieldInfo` property of the **CrystalReportViewer** control.

[Visual Basic]

```
Dim myParameterFields As ParameterFields =
myCrystalReportViewer.ParameterFieldInfo
```

[end]

[C#]

```
ParameterFields parameterFields =
crystalReportViewer.ParameterFieldInfo;
```

[end]

4. Call the `SetCurrentValuesForParameterField()` method, and pass in the ParameterFields instance, and the ArrayList instance.

[Visual Basic]

```
SetCurrentValuesForParameterField(myParameterFields, myArrayList)
```

[end]

[C#]

```
SetCurrentValuesForParameterField(parameterFields, arrayList);
```

[end]

You are now ready to build and run your project. It is expected that the report displays successfully because there is now code written to set current values into the parameter field.

To test the loading of the CustomersByCity report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The CustomersByCity report displays successfully, showing listings for customers in Paris and Tokyo.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you learn how to retrieve the default values from the parameter field and set those values into a ListBox control. These are used at the end of the tutorial to select new cities dynamically and filter the report based on those newly selected cities.

Create a ListBox Control that Displays Default Parameters

The remainder of the tutorial is concerned with displaying a complete list of default values for the parameter field in a ListBox control, and based on selections that you make from that ListBox control, refiltering the contents of the report.

In this section you learn how to populate the ListBox control from the default values of the parameter field.

Note Remember that you set the Default Values, a large list of cities, when you created this report at the beginning of the tutorial.

To do this, you must add and configure a ListBox control, and then create a helper method to populate the ListBox control.

To create and configure a ListBox control on the form

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   b) Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER.

      The CrystalReportViewer control drops by one line.
4. If you are developing a Windows project, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   b) From the **Properties** window, set **Dock** to "Bottom."
   c) Resize the **CrystalReportViewer** control, so that you leave enough room above it for a ListBox control.
   d) From the **Properties** window, set **Anchor** to "Top, Bottom, Left, Right."
5. From the **Toolbox**, drag a **ListBox** control above the **CrystalReportViewer** control.

   Note If a Smart Task appears on the ListBox (when using Visual Studio 2005), press Esc to close it.
6. Click on the **ListBox** control to select it.
7. From the **Properties** window, do the following:

   Set the **ID** or **Name** to "defaultParameterValuesList."

   Set the **SelectionMode** to "Multiple" (in a Windows project, "MultiExtended").
8. From the **File** menu, select **Save All**.

You are now ready to create a helper method that retrieves the default values from the parameter field.

To create a helper method that retrieves the default values from the parameter field

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a new private method named `GetDefaultValuesFromParameterField()` that returns an ArrayList instance, with ParameterFields passed into the method signature.

[Visual Basic]

```
Private Function GetDefaultValuesFromParameterField(ByVal
myParameterFields As ParameterFields) As ArrayList


End Function
```

[end]

[C#]

```
private ArrayList GetDefaultValuesFromParameterField(ParameterFields
parameterFields)
{
}
```

[end]

4. Retrieve the `ParameterField` instance from the `ParameterFields` indexed class, which is based on the index entry of the PARAMETER_FIELD_NAME constant.

[Visual Basic]

```
Dim myParameterField As ParameterField =
myParameterFields(PARAMETER_FIELD_NAME)
```

[end]

[C#]

```
ParameterField parameterField = parameterFields[PARAMETER_FIELD_NAME];
```

[end]

5. Retrieve a `ParameterValues` indexed class (as the variable `defaultParameterValues`) from the `DefaultValues` property of the `ParameterField` instance.

[Visual Basic]

```
Dim defaultParameterValues As ParameterValues =
myParameterField.DefaultValues
```

[end]

[C#]

```
ParameterValues defaultParameterValues = parameterField.DefaultValues;
```

[end]

6. Declare and instantiate an ArrayList.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

7. Create a `foreach` loop, to retrieve each `ParameterValue` instance from `defaultParameterValues`.

[Visual Basic]

```
For Each myParameterValue As ParameterValue In defaultParameterValues


Next
```

[end]

[C#]

```
foreach(ParameterValue parameterValue in defaultParameterValues)
{
```

```
    }
```
[end]

Within the `foreach` loop, you now create a nested conditional block that checks for discrete (as opposed to range) parameter values. Two versions of this conditional block exist, because the API has changed slightly across versions of Crystal Reports for Visual Studio. Check your API (using IntelliSense) to see which property is available under ParameterValue:

8. If the available property is IsRange then, within the `foreach` loop, enter this code:

[Visual Basic]
```
    If (Not myParameterValue.IsRange) Then


    End If
```
[end]
[C#]
```
    if(!parameterValue.IsRange)
    {
    }
```
[end]

9. Or, if the available property is Kind (DiscreteOrRangeKind, an enum with three values: DiscreteValue, RangeValue, DiscreteAndRangeValue) then, within the `foreach` loop, enter this code instead:

[Visual Basic]
```
    If (myParameterValue.Kind = DiscreteOrRangeKind.DiscreteValue) Then


    End If
```
[end]
[C#]
```
    if(parameterValue.Kind == DiscreteOrRangeKind.DiscreteValue)
    {
    }
```
[end]

10. Within this nested conditional block, cast the ParameterValue instance to its extended class, DiscreteParameterValue.

[Visual Basic]
```
    Dim myParameterDiscreteValue As ParameterDiscreteValue =
    CType(myParameterValue, ParameterDiscreteValue)
```
[end]
[C#]
```
    ParameterDiscreteValue parameterDiscreteValue =
    (ParameterDiscreteValue)parameterValue;
```
[end]

11. Also within the nested conditional block, add the Value property of the ParameterDiscreteValue instance (converted to String) into the ArrayList instance.

[Visual Basic]

```
myArrayList.Add(myParameterDiscreteValue.Value.ToString())
```

[end]

[C#]

```
arrayList.Add(parameterDiscreteValue.Value.ToString());
```

[end]

12. Outside the conditional block, and outside the `foreach` loop, at the end of the method, return the ArrayList instance from the method.

[Visual Basic]

```
Return myArrayList
```

[end]

[C#]

```
return arrayList;
```

[end]

You have retrieved the default values from the parameter field and returned them from the method as an ArrayList. You now bind this ArrayList to the defaultParameterValuesList ListBox control.

Your code varies slightly depending on whether you use a Web project or a Windows project; therefore, be sure to only complete either the Web or Windows procedure below.

To bind the ArrayList returned from the method to the ListBox in a Web project

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code immediately after the line of code that retrieves the ParameterFields instance.

   Within these line breaks, you can now enter additional code that sets the data source for the defaultParameterValuesList ListBox control when the page loads for the first time.

2. Within the line breaks, create a `Not IsPostBack` conditional block.

[Visual Basic]

```
If Not IsPostBack Then


End If
```

[end]

[C#]

```
if(!IsPostBack)
{
}
```

[end]

Note The `Not IsPostBack` conditional block is used to encapsulate code that should only be run the first time the page loads. Controls are typically bound to data values within `Not IsPostBack` conditional blocks so that their data values (and any subsequent control events) are not reset during page reloads.

3. Within the `Not IsPostBack` conditional block, set the DataSource property of the defaultParameterValuesList ListBox to the `GetDefaultValuesFromParameterField()` helper method, passing in the ParameterFields instance as a method parameter.

[Visual Basic]

```
defaultParameterValuesList.DataSource =
GetDefaultValuesFromParameterField(myParameterFields)
```

[end]

[C#]

```
defaultParameterValuesList.DataSource =
GetDefaultValuesFromParameterField(parameterFields);
```

[end]

   a) Still within the `Not IsPostBack` conditional block, call the `DataBind()` method of the defaultParameterValuesList ListBox.

[Visual Basic]

```
defaultParameterValuesList.DataBind()
```

[end]

[C#]

```
defaultParameterValuesList.DataBind();
```

[end]

To bind the ArrayList returned from the method to the ListBox in a Windows project

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code immediately after the line of code that retrieves the ParameterFields instance.

   Within these line breaks, you can now enter additional code that sets the data source for the defaultParameterValuesList ListBox control when the page loads for the first time.

2. Within the line breaks, set the DataSource property of the defaultParameterValuesList ListBox to the `GetDefaultValuesFromParameterField()` helper method, passing in the ParameterFields instance as a method parameter.

[Visual Basic]

```
defaultParameterValuesList.DataSource =
GetDefaultValuesFromParameterField(myParameterFields)
```

[end]

[C#]

```
defaultParameterValuesList.DataSource =
GetDefaultValuesFromParameterField(parameterFields);
```

[end]

You are now ready to build and run the project, to verify whether the defaultParameterValuesList ListBox is populated.

To test the population of the defaultParameterValuesList ListBox control

1. From the Build menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.

3. From the **Debug** menu, click **Start**.

   The defaultParameterValuesList ListBox control displays a complete list of default values (cities, in our tutorial).

4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you add a button to redisplay the report based on selections from the defaultParameterValuesList ListBox control.

Setting Parameters from ListBox Selections

In this section you add a button to redisplay the report based on selections from the defaultParameterValuesList ListBox control. Within the event method for this button, you call the same method that is called when the page first loads: `SetCurrentValuesForParameterField()`. But this time, rather than pass in arbitrary values (Paris and Tokyo), you pass in the selected values from the defaultParameterValuesList ListBox control.

To create and configure a redisplay Button on the form

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. From the **Toolbox**, drag a **Button** control to the right of the **ListBox** control.
4. Click on the **Button** control to select it.
5. From the **Properties** window, do the following:

   Set the **ID** or **Name** to "redisplay."

   Set the **Text** to "Redisplay Report."

You are now ready to create a button click event method that checks for selected items in the ListBox control, and passes them to `SetCurrentValuesForParameterField()` method.

Your code varies slightly for a Web project or a Windows project, so only complete either the Web or Windows procedure below.

To create the click event method for the redisplay Button in a Web project

1. Double-click the **Redisplay Report** button.

   You are taken to the code-behind class where a `redisplay_Click()` event method has been automatically generated.

2. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class for the System.Web.UI.WebControls namespace (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Web.UI.WebControls
```

[end]

[C#]

```
using System.Web.UI.WebControls;
```

[end]

Note This declaration is needed, to access the ListItem class.

3. Within the `redisplay_Click()` event method that has just been auto-generated, declare and instantiate an ArrayList.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

4. Create a `foreach` loop to retrieve each ListItem instance from the Items property of defaultParameterValuesList ListBox.

[Visual Basic]

```
For Each item As ListItem In defaultParameterValuesList.Items


Next
```

[end]

[C#]

```
foreach(ListItem item in defaultParameterValuesList.Items)
{
}
```

[end]

5. Within the `foreach` loop, create a nested conditional block that checks whether the Selected property of the current Item instance is set to True.

[Visual Basic]

```
If item.Selected Then


End If
```

[end]

[C#]

```
if(item.Selected)
{
}
```

[end]

6. Within the conditional block, add the Value property of the Item instance to the ArrayList instance.

[Visual Basic]

```
myArrayList.Add(item.Value)
```

[end]

[C#]

```
arrayList.Add(item.Value);
```

[end]

7. Outside the conditional block, and outside the `foreach` loop, rebind the file directory path of the CustomersByCity report to the ReportSource property of the CrystalReportViewer class.

   Note The file directory path that is shown below is for a Visual Studio 2005 project. "ProjectName" is replaced by the name of your Web Site. "UserName" is replaced by your logon name.

   The default path for a Web Site project:

[Visual Basic]

```
myCrystalReportViewer.ReportSource =
"C:\WebSites\ProjectName\CustomersByCity.rpt"
```

[end]

[C#]

```
crystalReportViewer.ReportSource =
"C:\\WebSites\\ProjectName\\CustomersByCity.rpt";
```

[end]

8. Retrieve the ParameterFields instance from the ParameterFieldInfo property of the CrystalReportViewer control.

[Visual Basic]

```
Dim myParameterFields As ParameterFields =
myCrystalReportViewer.ParameterFieldInfo
```

[end]

[C#]

```
ParameterFields parameterFields =
crystalReportViewer.ParameterFieldInfo;
```

[end]

9. Pass in the ParameterFields and ArrayList instance to the SetCurrentValuesForParameterField() method.

[Visual Basic]

```
SetCurrentValuesForParameterField(myParameterFields, myArrayList)
```

[end]

[C#]

```
SetCurrentValuesForParameterField(parameterFields, arrayList);
```

[end]

To create the click event method for the redisplay Button in a Windows project

1. Double-click the redisplay Button control.

   You are taken to the code-behind class where a `redisplay_Click()` event method has been automatically generated.

2. Within the redisplay_Click() event method that has just been auto-generated, declare and instantiate an ArrayList.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

3. Create a `foreach` loop, to retrieve each item (as a string) from the SelectedItems property of defaultParameterValuesList ListBox.

[Visual Basic]

```
For Each item As String In defaultParameterValuesList.SelectedItems


Next
```

[end]

[C#]

```
foreach(string item in defaultParameterValuesList.SelectedItems)
{
}
```

[end]

4. Within the `foreach` loop, add item string instance to the ArrayList instance.

[Visual Basic]

```
myArrayList.Add(item)
```

[end]

[C#]

```
arrayList.Add(item);
```

[end]

5. Outside the `foreach` loop, rebind the file directory path of the CustomersByCity report to the ReportSource property of the CrystalReportViewer class.

   Note The file directory path that is shown below is for a Visual Studio 2005 project. "ProjectName" is replaced by the name of your Windows project. "UserName" is replaced by your logon name.

   The default path for a Windows project:

[Visual Basic]

```
myCrystalReportViewer.ReportSource = "C:\Documents and
Settings\UserName\My Documents\Visual
Studio\Projects\ProjectName\CustomersByCity.rpt"
```

[end]

[C#]

```
crystalReportViewer.ReportSource = "C:\\Documents and
Settings\\UserName\\My Documents\\Visual Studio\\Projects\\
ProjectName\\CustomersByCity.rpt";
```

[end]

6. Retrieve the ParameterFields instance from the ParameterFieldInfo property of the CrystalReportViewer control.

[Visual Basic]

```
    Dim myParameterFields As ParameterFields =
    myCrystalReportViewer.ParameterFieldInfo
```

[end]

[C#]

```
    ParameterFields parameterFields =
    crystalReportViewer.ParameterFieldInfo;
```

[end]

7. Pass in the ParameterFields and ArrayList instance to the SetCurrentValuesForParameterField() method.

[Visual Basic]

```
    SetCurrentValuesForParameterField(myParameterFields, myArrayList)
```

[end]

[C#]

```
    SetCurrentValuesForParameterField(parameterFields, arrayList);
```

[end]

Now that the selected values from the ListBox control have been applied as the current values for the parameter field, you are ready to redisplay the report.

You are now ready to build and run the project, to verify that the parameter field has been reset successfully.

To test the population of the defaultParameterValuesList ListBox control

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, CTRL-click to select different cities in the list.
5. Click the **Redisplay Report** button.

The page reloads and displays the customer records for customers who live in the list of cities that you have selected.

6. Return to Visual Studio and click **Stop** to exit from debug mode.

Configuring Parameter Persistence

In this section, you configure persistence (in a Web-based tutorial) for the parameter field selections that are made from the ListBox control.

To demonstrate lack of persistence for parameter selections

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, SHIFT-click to select all cities in the list.
5. Click **Redisplay Report**.

The page reloads, and then displays the customer records for all customers in all cities. This is a large report that contains many pages.

6. On the CrystalReportViewer toolbar, click **Next Page**.

The list of selected cities is not persisted. Page 2 of the report is not displayed. Instead, you see the startup parameter settings again (Paris and Tokyo).

Note In Visual Studio .NET 2002, the list of selected cities may be persisted due to version differences.

7. Return to Visual Studio and click **Stop** to exit from debug mode.

You must add persistence code to your application so that changes made are persisted when Web pages are reloaded.

To begin, you add persistence code to the `ConfigureCrystalReports()` method, by adding an Else block to the If `Not IsPostBack` conditional block. You then set unique values for the ArrayList instance for either condition in the conditional block. On page startup, you set the default values ("Paris" and "Tokyo") into the ArrayList instance. On page reloads, you retrieve the ArrayList instance that is stored in Session.

To add persistence code to the ConfigureCrystalReports() method

1. In the `ConfigureCrystalReports()` method, cut and paste the two lines of code, which add Paris and Tokyo to the ArrayList, into the bottom of the `Not IsPostBack` conditional block.

   When you are finished, the conditional block should look like this:

[Visual Basic]

```
If Not IsPostBack Then
    defaultParameterValuesList.DataSource =
GetDefaultValuesFromParameterField(myParameterFields)
    defaultParameterValuesList.DataBind()
    myArrayList.Add("Paris")
    myArrayList.Add("Tokyo")
End If
```

[end]

[C#]

```
if (!IsPostBack)
{
   defaultParameterValuesList.DataSource =
GetDefaultValuesFromParameterField(parameterFields);
   defaultParameterValuesList.DataBind();
   arrayList.Add("Paris");
   arrayList.Add("Tokyo");
}
```

[end]

2. Add a final line of code to the conditional block that assigns the ArrayList instance to Session.

   Note You can use the variable name as the string identifier for the Session that you add.

[Visual Basic]

```
    Session("myArrayList") = myArrayList
```

[end]

[C#]

```
    Session["arrayList"] = arrayList;
```

[end]

3. Add an Else condition to the `Not IsPostBack` conditional block.
4. Within the Else block, retrieve the ArrayList instance from Session and cast it to ArrayList.

[Visual Basic]

```
    myArrayList = Ctype(Session("myArrayList"), ArrayList)
```

[end]

[C#]

```
    arrayList = (ArrayList)Session["arrayList"];
```

[end]

When you are finished, the conditional block should look like this:

[Visual Basic]

```
    If Not IsPostBack Then
        defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(myParameterFields)
        defaultParameterValuesList.DataBind()
        myArrayList.Add("Paris")
        myArrayList.Add("Tokyo")
        Session("myArrayList") = myArrayList
    Else
        myArrayList = Ctype(Session("myArrayList"), ArrayList)
    End If
```

[end]

[C#]

```
    if (!IsPostBack)
    {
       defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(parameterFields);
       defaultParameterValuesList.DataBind();
       arrayList.Add("Paris");
       arrayList.Add("Tokyo");
       Session["arrayList"] = arrayList;
    }
    else
```

```
    {
        arrayList = (ArrayList)Session["arrayList"];
    }
```

[end]

Those modifications to `ConfigureCrystalReports()` method ensure that the current instance of ArrayList is always available to be passed into the `SetCurrentValuesForParameterField()` method.

In the next section, you make two changes to the code in the button click event:

Take the ArrayList that you created and assign it to Session.

Replace the last two lines of code (that configure and display the report) with a call to the `ConfigureCrystalReports()` method to perform this functionality on a common set of code.

To modify the code in the button click event method to work with Session persistence

1. Delete the last three lines of code in the button click event method.

   The first line of code to delete is the code that binds the report file directory path to the ReportSource property of the CrystalReportViewer control.

   The second line of code to delete is the code that retrieves the ParameterFields instance from the ParameterFieldInfo property of the CrystalReportViewer control.

   The third line of code to delete is the call to the `SetCurrentValuesForParameterField()` method.

In the next step, you add two new lines of code to replace the deleted code.

2. Within the button click event method, immediately outside the `foreach` loop, add a line of code that assigns the ArrayList instance to Session.

Note You can use the variable name as the string identifier for the Session that you add.

[Visual Basic]

```
Session("myArrayList") = myArrayList
```

[end]

[C#]

```
Session["arrayList"] = arrayList;
```

[end]

3. Call the `ConfigureCrystalReports()` method.

This retrieves the ArrayList instance, applies it to the report, and binds the report to the CrystalReportViewer control.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

You are now ready to build and run the project, to verify that the parameter field has been reset successfully.

To test the population of the defaultParameterValuesList ListBox control

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, SHIFT-click to select all cities in the list.
5. Click **Redisplay Report**.

The page reloads and displays the customer records for all customers in all cities. This is a large report that contains many pages.

6. On the CrystalReportViewer toolbar, click **Next Page**.
7. The list of selected cities is now persisted. Page 2 of the report is displayed.
8. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully created a report with a discrete parameter value, created a helper method that can accept any set of values in a common format (an ArrayList), and applied those values to a report based on a specific parameter field name (the PARAMETER_FIELD_NAME constant).

You have also learned how to populate a ListBox control with the default values for a particular parameter field. You have learned how to retrieve the selected values from your ListBox and place them into an ArrayList, to recall the original helper method to change the list of parameter values and change the display of the report.

Note You may want to work further with this tutorial and add a subreport into your report, and then configure that subreport with a range parameter. If so, proceed to the next tutorial, which modifies your current tutorial.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_CRVObjMod_Parameters

C# Windows project: CS_Win_CRVObjMod_Parameters

Visual Basic Web Site: VB_Web_CRVObjMod_Parameters

Visual Basic Windows project: VB_Win_CRVObjMod_Parameters

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

CrystalReportViewer Object Model Tutorial:

Reading and Setting Range Parameters for a Subreport

Reading and Setting Range Parameters for a Subreport

Introduction

In the previous tutorial, Reading and Setting Discrete Parameters, you have learned how to create a report with a discrete parameter, and how to write code to set that parameter at runtime – both with hard-coded parameter values and parameter values that are passed from a ListBox or TextBox control.

In this tutorial, you learn how to add range parameters to a subreport.

You need to make four modifications to the project that you have created in Reading and Setting Discrete Parameters:

- You add a subreport into the original report.

  This subreport addresses the Orders table of the Xtreme database. The Orders table is related to the Customers table that is used in the previous tutorial by a Customer ID foreign key.

- You add a range parameter to the subreport that filters by a range of order dates.

- You add two Text controls to the form: orderStartDate, and orderEndDate, to set the order date range at runtime.

- You modify the SetCurrentValuesForParameter() method that you have created in the previous tutorial.

  This method creates a ParameterRangeValue instance that contains the startDate and endDate values, and then passes that ParameterRangeValue instance into the range parameter inside the subreport.

When you have completed this tutorial, you can filter the values that are displayed on the report at runtime. The code that you add limits the number of cities that are displayed in the main report, and limits the range of order dates to be displayed in the subreport.

This tutorial can also be completed with classes of the ReportDocument object model.

Adding a Subreport to the Original Report

To begin, you add a subreport to the original report.

To add a subreport

1. Open the project you created in the previous tutorial, Reading and Setting Discrete Parameters.
2. From **Solution Explorer**, double-click the **CustomersByCity** report to open it.
3. Right-click the **Details** gray bar and select Insert **Section Below**.
4. Right-click within the new **Details b** section that you have created, point to Insert, and then click **Subreport**.

A gray square appears around the mouse cursor.

5. Drag the gray rectangle into the new **Details b** section, and then click to release.
6. In the **Insert Subreport** dialog box, on the **Subreport** tab, select **Create a subreport with the Report Wizard**.

Note The **Insert Subreport** dialog box includes other options that allow you to choose an existing report and on-demand subreports.

7. In the **New report name** field, type "CustomerOrders."
8. Click **Report Wizard...**
9. In the **Available Data Sources** panel of the **Standard Report Creation Wizard** window, expand the **Create New Connection** folder.

Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to Crystal Reports Developer, the **Create New Connection** folder does not exist; the contents are shown at the top level.

10. From the subfolder that opens, expand the **ODBC (RDO)** folder.
11. In the **ODBC (RDO)** window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

The ODBC (RDO) folder expands and shows the Xtreme Sample Database.

12. Select the Orders table and click the > symbol to move the Orders table into the **Select Tables** panel, and then click **Next**.
13. From the **Available Fields** panel, select **Order ID**, **Order Date**, **Ship Date**, and **Ship Via**.
14. Click the **>** symbol to move these fields into the **Fields to Displa**y panel, and then click **Finish**.
15. In the **Insert Subreport** dialog box, select the **Link** tab.
16. In the panel **Container Report field(s)** to link to, in the list **Available fields**, expand the **Customers** table, select **Customer ID**, and then click the **>** symbol.
17. In the **Customers.Customer ID** field link panel that appears, leave the default selections unchanged.

These parameter and data selections auto-generate a relationship between the main report and the subreport.

18. Click **OK**.

The new subreport, CustomerOrders, is displayed within the **Details b** section of the **Main** report.

Note When you add a subreport to the Details section, the subreport displays for every row, which adds a performance cost to your report. If you do not need subreport information with that level of granularity, place the subreport in a Group section rather than a Details section.

You are now ready to verify the settings in the subreport.

To verify the settings in the subreport

1. In the report Details section, double-click on the CustomerOrders subreport to view it.

   At the bottom of the designer view, navigation buttons appear for both the Main Report and the CustomerOrders subreport.

2. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click Toggle Field View.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.

3. In the **Field Explorer**, expand **Parameter Fields**.
4. Verify that the parameter field **Pm-Customers.Customer ID** was auto-generated when the subreport was linked.
5. On the **Crystal Reports** toolbar, click **Select Expert**.
6. In the **Select Expert** dialog box, verify that the criteria **Orders.Customer ID** is equal to {Pm-Customers.Customer ID} is set, and then click **OK**.
7. From the **File** menu, select **Save All**.

   You have successfully added a CustomerOrders subreport to the CustomersByCity report. In the next section, you add an OrderDateRange parameter to the subreport.

To add an OrderDateRange parameter to the subreport

1. In the **Field Explorer**, right-click **Parameter Fields** and select **New...**
2. In the **Create Parameter Field** dialog box:

   Set the **Name** to "OrderDateRange."

   Set the **Prompting text** to "Specify a Date Range of Orders to display."

   Set the **Value type** to "**Date**."

   Set the **Options** to one selection only, "**Range value(s)**."
3. Click **OK**.
4. On the **Crystal Reports** toolbar, click **Select Expert**.
5. Click the **New** tab.
6. In the **Choose Field** dialog box, expand the **Orders** table, select **Order Date**, and then click **OK**.
7. On the new **Orders.OrderDate** tab, from the drop down criteria list, select formula:
8. Type the following formula:

```
{Orders.Order Date} in {?OrderDateRange}
```

9. Click **OK**.
10. From the **File** menu, select **Save All**.

    You have successfully added an OrderDateRange parameter to the subreport and linked it to the Orders.OrderDate column. In the next section, you add code to address the OrderDateRange parameter within the subreport.

Adding the Subreport Parameter Code

You are now ready to add the parameter code for the subreport to the code-behind class. To begin, you create a private helper method, SetDateRangeForOrders().

To create and code the SetDateRangeForOrders() method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the top of the class, add two new constants below the existing PARAMETER_FIELD_NAME constant added during the previous tutorial.

[Visual Basic]

```
Private Const SUBREPORT_PARAMETER_FIELD_NAME As String = "OrderDateRange"

Private Const SUBREPORT_NAME As String = "CustomerOrders"
```

[end]

[C#]

```
private const string SUBREPORT_PARAMETER_FIELD_NAME = "OrderDateRange";
private const string SUBREPORT_NAME = "CustomerOrders";
```

[end]

4. At the bottom of the class, create a new private method named `SetDateRangeForOrders()` with three parameters: `ParameterFields`, a string `startDate`, and a string `endDate`.

[Visual Basic]

```
Private Sub SetDateRangeForOrders(ByVal myParameterFields As
ParameterFields, ByVal startDate As String, ByVal endDate As String)


End Sub
```

[end]

[C#]

```
private void SetDateRangeForOrders(ParameterFields parameterFields,
string startDate, string endDate)
{
}
```

[end]

5. Within this method, declare and instantiate the `ParameterRangeValue` class.

[Visual Basic]

```
Dim myParameterRangeValue As ParameterRangeValue = New
ParameterRangeValue()
```

[end]

[C#]

```
ParameterRangeValue parameterRangeValue = new ParameterRangeValue();
```

[end]

Note For the ParameterRangeValue class to be accessible, you must include an `"Imports" [Visual Basic]` or `"using" [C#]` statement at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

6. Set the `StartValue` property of the `ParameterRangeValue` instance to the `startDate` method parameter.

[Visual Basic]

```
myParameterRangeValue.StartValue = startDate
```

[end]

[C#]

```
parameterRangeValue.StartValue = startDate;
```

[end]

Note The StartValue and EndValue properties of the ParameterRangeValue class accept values of type Object. This generic type allows the range value that is passed in to be of many types, including: text, number, date, currency, or time.

7. Set the `EndValue` property of the `ParameterRangeValue` instance to the `endDate` method parameter.

[Visual Basic]

```
myParameterRangeValue.EndValue = endDate
```

[end]

[C#]

```
parameterRangeValue.EndValue = endDate;
```

[end]

8. Set the lower and upper boundaries to be bound-inclusive.

[Visual Basic]

```
myParameterRangeValue.LowerBoundType = RangeBoundType.BoundInclusive
myParameterRangeValue.UpperBoundType = RangeBoundType.BoundInclusive
```

[end]

[C#]

```
parameterRangeValue.LowerBoundType = RangeBoundType.BoundInclusive;
parameterRangeValue.UpperBoundType = RangeBoundType.BoundInclusive;
```

[end]

Note For BoundInclusive, the upper and lower range values are included in the range.

You are now ready to assign the ParameterRangeValue instance to the parameter of the subreport.

9. Retrieve the `ParameterField` instance from the `ParameterFields` indexed class, which is based on two indexed values: the subreport parameter field name and the subreport name. Pass in the two constant values that you declared at the top of the class.

[Visual Basic]

```
Dim myParameterField As ParameterField =
myParameterFields(SUBREPORT_PARAMETER_FIELD_NAME, SUBREPORT_NAME)
```

[end]

[C#]

```
ParameterField parameterField =
parameterFields[SUBREPORT_PARAMETER_FIELD_NAME, SUBREPORT_NAME];
```

[end]

10. Call the `Clear()` method of `the CurrentValues` property of the `ParameterField` instance to remove any existing values from the `CurrentValues` property.

[Visual Basic]

```
myParameterField.CurrentValues.Clear()
```

[end]

[C#]

```
    parameterField.CurrentValues.Clear();
```

[end]

11. Add the `ParameterRangeValue` instance, which you created earlier, to the `CurrentValues` property of the `ParameterField` instance.

[Visual Basic]

```
    myParameterField.CurrentValues.Add(myParameterRangeValue)
```

[end]

[C#]

```
    parameterField.CurrentValues.Add(parameterRangeValue);
```

[end]

This step procedure has set start and end date values into a ParameterRangeValue instance and passed those values to the OrderDateRange parameter in the CustomerOrders subreport.

Adding TextBox Controls to Hold Range Parameter Values

In this section, you add two TextBox controls to provide start and end date values at runtime to the OrderDateRange range parameter in the CustomerOrders subreport.

Note If you implement this tutorial in a Web Site, the persistence of date values that users enter into the text boxes are maintained by ViewState.

To create and configure a redisplay Button on the form

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:
4. Click between the **ListBox** control and the **Button** control.
5. Press ENTER three times to create two rows between the ListBox control and the Button control.
6. In the first row created below the **ListBox** control, type "Order Start Date."
7. In the second row created below the **ListBox** control, type "Order End Date."
8. If you are developing a Windows project, do the following:
9. From the **Toolbox**, drag two **Label** controls to the right of the ListBox control. Place one label above the other, with both of them above the **Button** control.
10. Select the first **Label** control, and then from the **Properties** window, set the **Text** property to "Order Start Date."
11. Select the second **Label** control, and then from the **Properties** window, set the **Text** property to "Order End Date."

The remaining steps apply to both Web and Windows projects.

12. From the **Toolbox**, drag a TextBox control to the right of "Order Start Date."
13. Click on the **TextBox** control to select it.
14. From the **Properties** window, set the **ID** (or **Name**) to "orderStartDate."
15. From the **Toolbox**, drag a **TextBox** control to the right of "Order End Date."

16. Click on the **TextBox** control to select it.
17. From the **Properties** window, set the **ID** (or **Name**) to "orderEndDate."
18. From the **File** menu, select **Save All**.

Modifying Methods to Call the Subreport

You must now modify the `ConfigureCrystalReports()` method and the `redisplay_Click()` event method to receive information from these TextBox controls and apply them to the `SetDateRangeForOrders()` method, to have the parameter information processed for subreports.

In the previous tutorial, Reading and Setting Discrete Parameters, you designed these methods in two different ways, depending on whether you included Session persistence.

> **Note** Windows projects do not require Session persistence. Web Sites typically require Session persistence.

Choose from one (but not both) of the step procedures below. Either modify the methods that exclude Session persistence, or modify the methods that include Session persistence:

Modifying Methods that Exclude Session Persistence.

Modifying Methods that Include Session Persistence.

Modifying Methods that Exclude Session Persistence

If you created the previous tutorial Reading and Setting Discrete Parameters and excluded Session persistence, work through the following procedures. Otherwise go instead to Modifying the Methods that Include Session Persistence.

To modify the ConfigureCrystalReports() method that excludes Session persistence

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the lines which assign "Paris" and "Tokyo" as ArrayList variables.
2. Within the line breaks, declare and set hard-coded values for two string variables, `startDate` and `endDate`.

[Visual Basic]

```
Dim startDate As String = "8/1/1997"
Dim endDate As String = "8/31/1997"
```

[end]

[C#]

```
string startDate = "8/1/1997";
string endDate = "8/31/1997";
```

[end]

3. Below the code that calls `SetCurrentValuesForParameterField()`, call the `SetDateRangeForOrders()` method and pass in the `ParameterFields` instance and the `startDate` and `endDate` variables.

[Visual Basic]

```
SetDateRangeForOrders(myParameterFields, startDate, endDate)
```

[end]

[C#]

```
SetDateRangeForOrders(parameterFields, startDate, endDate);
```

[end]

4. From the File menu, select Save All.

Next, you modify the redisplay_Click event method.

To modify the redisplay_Click() method that excludes Session persistence

1. In the `redisplay_Click()` event method, create a couple of line breaks in the code above the line that binds the report to the **CrystalReportViewer** control.
2. Within the line breaks, declare and set values for two string variables, `startDate` and `endDate`, from the **TextBox** controls that you added to the Web or Windows form.

[Visual Basic]

```
Dim startDate As String = orderStartDate.Text
Dim endDate As String = orderEndDate.Text
```

[end]

[C#]

```
string startDate = orderStartDate.Text;
string endDate = orderEndDate.Text;
```

[end]

3. Below the code that calls `SetCurrentValuesForParameterField()`, call the `SetDateRangeForOrders()` method and pass in the `ParameterFields` instance and the `startDate` and `endDate` variables.

[Visual Basic]

```
SetDateRangeForOrders(myParameterFields, startDate, endDate)
```

[end]

[C#]

```
SetDateRangeForOrders(parameterFields, startDate, endDate);
```

[end]

4. From the **File** menu, select **Save All**.

Modifying Methods that Include Session Persistence

If you created the previous tutorial Reading and Setting Discrete Parameters and included Session persistence, work through the following procedures. Otherwise, continue to Modifying the Methods that Exclude Session Persistence.

To modify the ConfigureCrystalReports() method that includes Session persistence

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the line that declares and instantiates ArrayList.
2. Within the line breaks, declare two string variables, `startDate` and `endDate`.

[Visual Basic]

```
Dim startDate As String
```

```
    Dim endDate As String
```
[end]
[C#]
```
    string startDate;
    string endDate;
```
[end]

3. Within the `Not IsPostBack` conditional block, enter default values for the `startDate` and `endDate` variables.

[Visual Basic]
```
    startDate = "8/1/1997"
    endDate = "8/31/1997"
```
[end]
[C#]
```
    startDate = "8/1/1997";
    endDate = "8/31/1997";
```
[end]

4. Within the `Not IsPostBack` conditional block, assign the `startDate` and `endDate` variables into `Session`.

[Visual Basic]
```
    Session("startDate") = startDate
    Session("endDate") = endDate
```
[end]
[C#]
```
    Session["startDate"] = startDate;
    Session["endDate"] = endDate;
```
[end]

5. Within the `Else` block, after the ArrayList instance is retrieved from Session, retrieve the `startDate` and `endDate` variables from `Session`.

[Visual Basic]
```
    startDate = Session("startDate").ToString()
    endDate = Session("endDate").ToString()
```
[end]
[C#]
```
    startDate = Session["startDate"].ToString();
    endDate = Session["endDate"].ToString();
```
[end]

   With that approach, you reach the end of the block with the date variables assigned in either case. This follows parallel logic to the assignment of the ArrayList variable that you configured in the previous tutorial.

6. Below the code that calls `SetCurrentValuesForParameterField()`, call the `SetDateRangeForOrders()` method and pass in the `ParameterFields` instance and the `startDate` and `endDate` variables.

[Visual Basic]

```
SetDateRangeForOrders(myParameterFields, startDate, endDate)
```

[end]

[C#]

```
SetDateRangeForOrders(parameterFields, startDate, endDate);
```

[end]

7. From the **File** menu, select **Save All**.

Next, you modify the redisplay_Click event method.

To modify the redisplay_Click() method that includes Session persistence

8. In the `redisplay_Click()` event method, create a couple of line breaks in the code after the line that assigns the ArrayList instance to Session.
9. Within the line breaks, assign the Text property of the `orderStartDate TextBox` and the `orderEndDate` TextBox to `Session` variables.

[Visual Basic]

```
Session("startDate") = orderStartDate.Text
Session("endDate") = orderEndDate.Text
```

[end]

[C#]

```
Session["startDate"] = orderStartDate.Text;
Session["endDate"] = orderEndDate.Text;
```

[end]

10. From the **File** menu, select **Save All**.

Those startDate and endDate Session values are now retrieved and applied when the `ConfigureCrystalReports()` method is called.

You are now ready to build and run the project, to verify that the TextBox values are resetting the range parameter in the subreport.

Testing the Subreport Parameter

You are now ready to test the setting of the subreport parameter from the TextBox values.

To test the setting of the subreport parameter

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, CTRL-click to select at least four different cities in the list.
5. In the **startDate** TextBox control, enter "1/1/1997."
6. In the **endDate** TextBox control, enter "8/31/1997."
7. Click the **Redisplay Report** button.

The page reloads and displays the customer records for customers who live in the list of cities that have just been selected, as well as a subreport that displays orders for the date range specified above.

8. In the **CrystalReportViewer** control, increase the **Zoom** level to 125%.
9. The page reloads at 125% zoom. The values that are selected for both cities and order date range are persisted.
10. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully modified your tutorial project to use a report containing a subreport, and set an order date range to the range parameter that is created in the subreport.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_CRVObjMod_ParametersSubrpt

C# Windows project: CS_Win_CRVObjMod_ParametersSubrpt

Visual Basic Web Site: VB_Web_CRVObjMod_ParametersSubrpt

Visual Basic Windows project: VB_Win_CRVObjMod_ParametersSubrpt

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

CrystalReportViewer Object Model Tutorial:

Filtering Data Using Selection Formulas

Filtering Data Using Selection Formulas

Introduction

Selection formulas are used to filter records that you want to display on a Crystal report. To write selection formulas, you can use the Basic syntax and the Crystal syntax. For more information about how to write selection formulas, see Appendix: Formula Reference.

In this tutorial, you create a selection formula to filter customer records where the Last Year's Sales field is greater than a specific value, and the Customer Name field is compared to another string. A ListBox control is used to select a comparison operator for the Customer Name field. You can choose to display customer names that are equal to, less than, greater than, less than or equal to, greater than or equal to, or not equal to the string value that you have specified.

The formula is passed as a string variable to the SelectionFormula property of the CrystalReportViewer class. Once the property is set, the Crystal report that binds to the CrystalReportViewer control is filtered before it is displayed.

Creating a Report with a Selection Formula

In this section, you create a report that draws its information from the sample Microsoft Access database that ships with Crystal Reports.

To create a report with secure data from the Northwind database

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In Solution Explorer, right-click the project name that is in bold type, point to Add, and then click Add New Item.
2. In the Add New Item dialog box, in the Templates view, select the Crystal Report template.
3. In the Name field, enter the name "CustomersBySalesName.rpt" and click Add.

   Note In Visual Studio .NET 2002 or 2003, the button is named Open.

   If you have not registered before, you are asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
4. In the Create New Crystal Report Document panel of the Crystal Reports Gallery dialog box, select Using a Report Wizard.
5. In the Choose an Expert panel, select Standard, and then click OK.
6. In the Available Data Sources panel of the Standard Report Creation Wizard window, expand the Create New Connection folder.

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to the full version, the Create New Connection folder does not exist; the contents are shown at the top level.
7. From the subfolder that opens, expand the ODBC (RDO) folder.

8. In the ODBC (RDO) window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click Finish.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
9. Expand the Tables node, and then double-click the Customer table to move the table to the Selected Tables panel, and then click Next.
10. Expand the Customer table, and then CTRL-click Customer Name, and Last Year's Sales.
11. Click the > symbol to move these fields into the Fields to Display panel, and then click Next.
12. In the Available Fields panel, under Report Fields, double-click Customer.Customer Name to move the field into the Group By panel, and then click Finish.

    The CustomersBySalesName report is created and loaded into the main window of Visual Studio.

Next, you create a selection formula to filter the data based on the Last Year's Sales field.

To create a selection formula based on Last Year's Sales

1. Open the Web or Windows Form.

a) From the View menu, click Code to view the code-behind class for this Web or Windows Form.

b) For a Web Site, within the ConfigureCrystalReports() method (that you have created in Appendix: Project Setup), create a Not IsPostBack conditional block.

[Visual Basic]

```
If Not IsPostBack Then


End If
```

[end]

[C#]

```
if(!IsPostBack)
{
}
```

[end]

Note The `Not IsPostBack` conditional block encapsulates code that should be run only the first time the page loads.

2. For a Web Site, add the following lines of code within the `Not IsPostBack` conditional block.

   For a Windows project, add the code to the `ConfigureCrystalReports()` method, without the `Not IsPostBack` conditional block.
3. Enter the formula to select only records with Last Year's Sales greater than $11000.00, and customer names that begin with the letter "A".

[Visual Basic]

```
Dim mySelectFormula As String = "{Customer.Last Year's Sales} > 11000.00
" _
```

```
        & "AND Mid({Customer.Customer Name}, 1, 1) = ""A"" "
```

[end]

[C#]

```
        string selectFormula = "{Customer.Last Year's Sales} > 11000.00 "
        + "AND Mid({Customer.Customer Name}, 1, 1) = \"A\"";
```

[end]

4. Assign the selection formula string to the SelectionFormula property of the CrystalReportViewer control.

[Visual Basic]

```
        myCrystalReportViewer.SelectionFormula = mySelectFormula
```

[end]

[C#]

```
        crystalReportViewer.SelectionFormula = selectFormula;
```

[end]

Binding the Report

When you followed the instructions in the section Appendix: Project Setup to prepare for this tutorial, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous steps, you added a CustomersBySalesName report and a selection formula to the project.

In this section, you bind the file directory path of the CustomersBySalesName report to the CrystalReportViewer control. Then you test whether the report displays correctly with the records that are filtered by the selection formula.

To bind the file directory path of the CustomersBySalesName report to the CrystalReportViewer control

1. Open the Web or Windows Form.

a) From the View menu, click Code to view the code-behind class for this Web or Windows Form.

b) Locate the ConfigureCrystalReports() method (that you have created in Appendix: Project Setup).

c) Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
        Dim reportPath As String = Server.MapPath("CustomersBySalesName.rpt")
```

[end]

[C#]

```
        string reportPath = Server.MapPath("CustomersBySalesName.rpt");
```

[end]

For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"CustomersBySalesName.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"CustomersBySalesName.rpt";
```

[end]

2. Assign the file directory path of the NorthwindCustomers report to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = reportPath
```

[end]

[C#]

```
crystalReportViewer.ReportSource = reportPath;
```

[end]

To test the selection formula for the CustomersBySalesName report

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the \bin\ [Visual Basic] or \bin\debug\ [C#] subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the Debug menu, click Start.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The Crystal report displays four customer records: Alley Cat Cycles, Ankara Bicycle Company, Arsenault et Maurier, and Athens Bicycle Co.
5. Return to Visual Studio and click Stop to exit from debug mode.

Adding Controls to Use in the Selection Formula

In this section, you add controls to dynamically change the values that are used in the selection formula. For the Last Year's Sales field, you add a textbox to specify the

minimum value of sales to be displayed in the Crystal report. For the Customer Name field, you add a DropDownList and TextBox control to specify what customer names to display.

To add controls to use in the selection formula

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   b) Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER three times.

The **CrystalReportViewer** control drops by three lines.

4. If you are developing a Windows project, do the following:
   a) Click the CrystalReportViewer control to select it.
   b) From the Properties window, set Dock to "Bottom".
   c) Resize the **CrystalReportViewer** control, so that approximately three lines appear above it.
   d) From the **Properties** window, set **Anchor** to "Top, Bottom, Left, Right".
5. If you are developing a Web Site, on the first line type, "Enter the minimum value for last year's sales: $".
6. If you are developing a Windows project, do the following:
   a) From the Toolbox, drag a Label control to the top of the form.
   b) From the Properties window, set the Text property to "Enter the minimum value for last year's sales: $".
   c) From the **Toolbox**, drag a **TextBox** control to the right of the text.
   d) Select the **TextBox** control, and then from the **Properties** window, do the following:

      Set the **ID** (or **Name**) to "lastYearsSales".

      Set the **Text** to "11000.00".
   e) If you are developing a Web Site, on the second line, type "Display the customer names".
7. If you are developing a Windows project, do the following:
   a) From the Toolbox, drag a Label control to the second line of the form.
   b) From the Properties window, set the Text property to "Display the customer names".
   c) From the **Toolbox**, drag a **DropDownList** (**ComboBox** in a Windows project) to the right of the text.
   d) Select the **DropDownList** (**ComboBox**) control, and then from the **Properties** window, set the **ID** (or **Name**) to "selectOperatorList".
   e) From the **Toolbox**, drag a **TextBox** control to the right of the DropDownList control.
   f) Select the **TextBox** control, and then from the **Properties** window, do the following:

Set the **ID** (or **Name**) to "customerName".

Set the **Text** to "A".

g) From the Toolbox, drag a Button control to the third line of the form, and above the CrystalReportViewer control.

h) Select the **Button** control, and then from the **Properties** window, do the following:

Set the **ID** (or **Name**) to "redisplay".

Set the **Text** to "Redisplay Report".

Setting the Selection Formula Manually in Code

You are now ready to add code to modify the selection formula in the code-behind class.

To code the selection formula

1. Open the Web or Windows Form.
2. From the View menu, select Designer.
3. Double-click Redisplay Report.

   The code-behind class for the report appears and shows that a `redisplay_Click()` event method has been automatically generated.
4. For the selection formula, create a string variable that takes the values from the TextBox controls.

   The selection formula is similar to the text you have typed within the `ConfigureCrystalReports()` method. Instead of a minimum sales value of $11000.00, you use the value from the lastYearsSales TextBox control. For the Customer Name field, you use the value from the customerName TextBox control.

[Visual Basic]

```
Dim mySelectFormula As String = "{Customer.Last Year's Sales} > " &
lastYearsSales.Text _
& " AND Mid({Customer.Customer Name}, 1) > """ & customerName.Text & """"
```

[end]

[C#]

```
string selectFormula = "{Customer.Last Year's Sales} > " +
lastYearsSales.Text
+ " AND Mid({Customer.Customer Name}, 1) > \"" + customerName.Text + "\"";
```

[end]

5. Assign the string variable to the SelectionFormula property of the CrystalReportViewer control.

[Visual Basic]

```
    myCrystalReportViewer.SelectionFormula = mySelectFormula
```

[end]

[C#]

```
    crystalReportViewer.SelectionFormula = selectFormula;
```

[end]

6. Rebind the CustomerBySalesName report to the ReportSource property of the CrystalReportViewer control.

   Note The file directory path that is shown here is for a Visual Studio 2005 project. "ProjectName" is replaced by the name of your Web or Windows project. "UserName" is replaced by your computer logon name.

   The default path for a Web Site project is as follows:

[Visual Basic]

```
myCrystalReportViewer.ReportSource =
"C:\WebSites\ProjectName\CustomersBySalesName.rpt"
```

[end]

[C#]

```
crystalReportViewer.ReportSource =
"C:\\WebSites\\ProjectName\\CustomersBySalesName.rpt";
```

[end]

   The default path for a Windows project is as follows:

[Visual Basic]

```
myCrystalReportViewer.ReportSource = "C:\Documents and
Settings\UserName\My Documents\Visual
Studio\Projects\ProjectName\CustomersBySalesName.rpt"
```

[end]

[C#]

```
crystalReportViewer.ReportSource = "C:\\Documents and
Settings\\UserName\\My Documents\\Visual Studio\\Projects\\
ProjectName\\CustomersBySalesName.rpt";
```

[end]

You have now created a selection formula that you can modify at run time.

To test the selection formula

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.
4. In the lastYearsSales TextBox, type "200000".
5. In the customerName TextBox, type "SAB".
6. Click Redisplay Report.

   The Crystal report displays three customer records: SAB Mountain, Tek Bikes, and Tienda de Bicicletas El Pardo.

   Only customer records with names greater than "SAB" and last year's sales greater than "200000" are displayed.
7. Return to Visual Studio and click Stop to exit from debug mode.

Using a DropDownList Control to Modify the Selection Formula

In this section, you populate the DropDownList control with comparison operators. You create an enum that contains the comparison operators.

The DropDownList control selects whether you want to display customer names that are equal to, less than, greater than, less than or equal to, greater than or equal to or not equal to the text that you have entered into the TextBox control.

In the redisplay_Click() event method, you modify the string that is currently assigned to the SelectionFormula property of the CrystalReportViewer control.

To create the CeComparisonOperator enum

1. In Solution Explorer, right-click the project name that is in bold type, point to Add, and then click Add New Item.
2. In the Add New Item dialog box, select Class from the Templates view.
3. In the Name field, type "CeComparisonOperator", and then click Add.

   Note In Visual Studio 2005, you may be asked to place this class in a Code directory. Click the Yes button.
4. In the class signature, change the word class to enum to convert the class to an enumeration.

   In a C# Windows project for Visual Studio 2005, you also must change the namespace to the name of your project.

   Note In Visual Basic, remember to change both the opening and the closing signatures of the class to enum.
5. Because enums do not have constructors, delete the default constructor method that is provided in the C# version of the code.
6. Inside the enum, enter the values:

[Visual Basic]

```
    EqualTo
    LessThan
    GreaterThan
    LessThan_or_EqualTo
    GreaterThan_or_EqualTo
    Not_EqualTo
```

[end]

[C#]

```
    EqualTo,
    LessThan,
    GreaterThan,
    LessThan_or_EqualTo,
    GreaterThan_or_EqualTo,
```

```
    Not_EqualTo
```

[end]

The following procedures explain how to bind the CeComparisonOperator enum to the DropDownList control for a Web Site or for a Windows project. Complete the instructions in one of the step procedures below.

To populate the DropDownList control from the CeComparisonOperator enum for a Web Site

1. Open the Web Form.
2. From the View menu, click Code.
3. Within the Not IsPostBack conditional block of the ConfigureCrystalReports() method, before the selection formula string declaration, set the DataSource property of the DropDownList control to the values of the CeComparisonOperator enum.

[Visual Basic]

```
    selectOperatorList.DataSource =
    System.Enum.GetValues(GetType(CeComparisonOperator))
```

[end]

[C#]

```
    selectOperatorList.DataSource =
    System.Enum.GetValues(typeof(CeComparisonOperator));
```

[end]

4. Now call the DataBind() method of the selectOperatorList DropDownList to bind the values to the control.

[Visual Basic]

```
    selectOperatorList.DataBind()
```

[end]

[C#]

```
    selectOperatorList.DataBind();
```

[end]

To populate the DropDownList control from the CeComparisonOperator enum for a Windows project

1. Open the Windows Form.
2. From the View menu, click Code.
3. Within the ConfigureCrystalReports() method, before the selection formula string declaration, set the DataSource property of the selectOperatorList ComboBox to the values of the CeComparisonOperator enum.

[Visual Basic]

```
    selectOperatorList.DataSource =
    System.Enum.GetValues(GetType(CeComparisonOperator))
```

[end]

[C#]

```
    selectOperatorList.DataSource =
    System.Enum.GetValues(typeof(CeComparisonOperator));
```

[end]

Next, you create the GetSelectedCompareOperator() helper method to return the selected index as a string that represents a comparison operator sign.

To create the GetSelectedCompareOperator() helper method

1. At the bottom of the class, create a private helper method named GetSelectedCompareOperator() that returns a string variable.

[Visual Basic]

```
Private Function GetSelectedCompareOperator() As String


End Function
```

[end]

[C#]

```
private string GetSelectedCompareOperator()
{
}
```

[end]

2. Within the method, create a "Select Case" [Visual Basic] or "switch" [C#] statement that references the members of the CeComparisonOperator enum, and returns the comparison operator sign as a string variable.

[Visual Basic]

```
Select Case selectOperatorList.SelectedIndex
   Case CeComparisonOperator.EqualTo
         return "="
   Case CeComparisonOperator.LessThan
         return "<"
   Case CeComparisonOperator.GreaterThan
         return ">"
   Case CeComparisonOperator.LessThan_or_EqualTo
         return "<="
   Case CeComparisonOperator.GreaterThan_or_EqualTo
         return ">="
   Case CeComparisonOperator.Not_EqualTo
         return "<>"
   Case Else
         return "="
End Select
```

[end]

[C#]

```
switch ((CeComparisonOperator)selectOperatorList.SelectedIndex)
{
   case CeComparisonOperator.EqualTo:
         return "=";
   case CeComparisonOperator.LessThan:
         return "<";
   case CeComparisonOperator.GreaterThan:
         return ">";
   case CeComparisonOperator.LessThan_or_EqualTo:
         return "<=";
   case CeComparisonOperator.GreaterThan_or_EqualTo:
         return ">=";
   case CeComparisonOperator.Not_EqualTo:
         return "<>";
   default:
         return "=";
}
```

[end]

In the redisplay_Click() event method, a "greater than" sign (">") is currently used for the Customer Name field selection. Next, you learn how to change the sign to the comparison operator that you have selected from the DropDownList control. The selected sign is returned as a string when you call the GetSelectedCompareOperator() helper method.

To modify the Customer Name's comparison operator that is assigned to the SelectionFormula property

1. At the top of the redisplay_Click() event method, call the GetSelectedCompareOperator() helper method, and assign the result to a string variable.

[Visual Basic]

```
Dim mySelectedOperator As String = GetSelectedCompareOperator()
```

[end]

[C#]

```
string selectedOperator = GetSelectedCompareOperator();
```

[end]

2. For the selection formula string variable, replace the "greater than" sign (">") with the selected operator string.

[Visual Basic]

```
Dim mySelectFormula As String = "{Customer.Last Year's Sales} > " &
lastYearsSales.Text _
```

```
    & " AND Mid({Customer.Customer Name}, 1) " & mySelectedOperator & " """ &
    customerName.Text & """"
[end]
[C#]
    string selectFormula = "{Customer.Last Year's Sales} > " +
    lastYearsSales.Text
    + " AND Mid({Customer.Customer Name}, 1) " + selectedOperator + " \"" +
    customerName.Text + "\"";
[end]
```

You have created a selection formula that depends on the values that you enter for the Last Year's Sales field, and the Customer Name field.

You can now build and test the selection formula.

To test the selection formula for the CustomersBySalesName report

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.
   a. In the lastYearsSales TextBox, type "40000".
   b. In the customerName TextBox, type "Athens Bicycle Co."
   c. In the DropDownList, select "LessThan."
   d. Click **Redisplay Report**.

      The Crystal report displays two customer records: Alley Cat Cycles, and Ankara Bicycle Company.
4. Return to Visual Studio and click Stop to exit from debug mode.

Conclusion

You have successfully created a selection formula to modify the records that are displayed on the Crystal report. You have learned to read values from TextBox and DropDownList controls to modify the selection formula. You have also learned to create an enum of comparison operators that allows you to select how you want to filter the data.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_CRVObjMod_FilteringData

C# Windows project: CS_Win_CRVObjMod_FilteringData

Visual Basic Web Site: VB_Web_CRVObjMod_FilteringData

Visual Basic Windows project: VB_Win_CRVObjMod_FilteringData

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

CrystalReportViewer Object Model Tutorial:

Customizing the CrystalReportViewer Control

Customizing the CrystalReportViewer Control

Introduction

In this tutorial, you learn how to customize the look of the CrystalReportViewer control through use of the properties from its underlying class.

You also learn how to use the methods for page selection, zoom, search, and print.

To begin, you learn how to customize the CrystalReportViewer toolbar. You need a ListBox that stores the properties that are available for the toolbar. Only the properties selected from the ListBox control are displayed on the CrystalReportViewer toolbar.

Then, you add a second ListBox to store the elements for the report. For a Web Site, you also choose to display all the report pages as a single page or as separate pages.

You learn how to customize the background color through a DropDownList control.

Next, you learn how to select the report page that you want to view. You need a TextBox control to enter the page number, and a Button control to reload the report to your selected page. You also need a TextBox and a Button control to modify the zoom factor and to search for text in your report.

For a Web Site, you have access to properties of the CrystalReportViewer control that are not available in a Windows project: one property to choose the print mode and other properties to change the width, style, and color of borders.

Creating a Customized Settings Table

In this section, you create and configure a table (in a Web Site) or a TableLayoutPanel control (in a Windows project) to hold the various controls that make up your customized settings table.

Because Web Sites and Windows projects each use a different kind of table, please select the step procedure that corresponds to your Web Site or Windows project.

To create a customized settings table for a Web Site

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. Open the Default.aspx page (the Web form) in Design view.
2. Click the CrystalReportViewer control to select it.
3. Press the left arrow to move the cursor to the left of the CrystalReportViewer control and then press Enter.
4. Press the up arrow to move the cursor to the empty line above the CrystalReportViewer control.
5. From the Layout menu, click Insert Table.
6. In the Insert Table dialog box, select the Custom radio button.
7. Note The Custom radio button is already selected by default.
8. In the Layout panel, select the Width checkbox and leave the value at 100%.

9. Increase the Rows count to 6 and the Columns count to 4.
10. In the Attributes panel, select the Border checkbox, and increase the count to 1.
11. Click the Cell Properties... button.
12. In the Cell Properties dialog box, in the Layout panel, set the Vertical align combo box to Top.
13. Select the No Wrap checkbox, and then click OK.
14. Click OK again to close the Insert Table dialog box.

You are now ready to add customized controls into this table for your Web Site.

To create a customized settings table for a Windows project

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

Open the Windows Form in **Design** view.

1. Click the Form title bar to select the entire form, and then drag the lower right corner of the form to enlarge it to fill the main area.
2. Click the **CrystalReportViewer** control to select it.
3. From the **Properties** window, set **Dock** to "Bottom."
4. From the **Properties** window, set **Anchor** to "Top, Bottom, Left, Right."
5. From the **Toolbox**, drag a **TableLayoutPanel** control to the upper left of the Windows Form.

   A TableLayoutPanel control is displayed, showing two columns and two rows.
6. If the Smart Task is not open, click the triangular button on the upper-right corner of the **TableLayoutPanel** control.

   The Smart Task panel named "TableLayoutPanel Tasks" opens.
7. In the **TableLayoutPanel Tasks** tag, click the **Edit Rows and Columns** link.
8. In the **Column and Row Styles** dialog box, in the **Member Type** combo box, select **Columns**.
9. Click **Add** until you have a total of four columns.
10. For each column, do the following:
11. Select the column.
12. In the **Size Type** panel select **Percent**.
13. Set each value to 25%.
14. In the **Member Type** combo box, select **Rows**.
15. Click **Add** until you have a total of five rows.

    Note The table for the Windows project requires one less row than the table for a Web Site, because there are a few less configurable options on a CrystalReportViewer control for a Windows project.
16. For each row, do the following:
17. Select the row.

18. In the **Size Type** panel select **Percent**.
19. Set the value of the first row to 40%, and for each subsequent row set the value to 15%.

    Note (1 x 40%) and (4 x 15%) = 100% of available space.
20. Click **OK**.
21. Close the **TableLayoutPanel** tasks tag.
22. Drag the bottom right corner of the **TableLayoutPanel** control to enlarge the table until it fills the space you have created above the **CrystalReportViewer** control.

You are now ready to add customized controls into this table for your Windows project.

Report and Toolbar Elements of the CrystalReportViewer control

In this tutorial you will manipulate the various Report and Toolbar Elements of the CrystalReportViewer control.

Viewer elements

The default elements for the CrystalReportViewer control vary slightly for Web Sites or Windows projects:

For both Web and Windows:

Toolbar: displays a toolbar above the main area of the report. Individual elements within the toolbar are controlled separately.

Note For more information, see the section on Toolbar elements below.

Group tree: displays the headings for each group in the report, similar to a directory tree; it appears on the left column panel of the report.

For Web only:

Main page: displays the report in the main area of the page.

Enable separate pages: determines whether to display the report in a single Web page or as separate formatted pages.

For Windows only:

Status bar: displays current page number and other information about the report, at the bottom of the report area.

Toolbar elements

The default elements for the toolbar vary slightly for Web Sites or Windows projects:

For both Web and Windows:

Group tree button: shows or hides the group tree section of the report.

Export: saves the Crystal report to another file format, such as RPT, PDF, DOC, XLS, or RTF files.

Print: prints the Crystal report to a PDF file, or it calls the Print dialog box.

Page navigation: allows you to select next, previous, last, or first page to view.

Go to page: allows you to type in the page number you want to view.

Search: allows you to type in a string you want to search for in the report.

Zoom factor: allows you to select the zoom factor for the report.

For Web only:

View list (only for a Web Site): chooses which view of the report to display, (for example, subreports and so on).

Drill up: opens a page that has more specific information than the current topic.

Crystal logo: displays the Crystal Reports product logo.

For Windows only:

Refresh: redisplays the report.

Close current view: closes the current view of the report if more than one view is open.

Adding a Hide Show Mechanism for Report and Toolbar Elements

In this section, you learn how to add a hide show mechanism to determine which elements to display on the CrystalReportViewer toolbar.

You begin by adding the ListBox and Button controls into the table on the Web or Windows form.

You then create two enums that list the report elements and the toolbar elements, and you then populate each ListBox with the values from one of the enums.

Next, you code the Button control's click event to update the report and toolbar elements.

Within the event handler the properties of the CrystalReportViewer class are set based on selections in the two ListBox controls. If an item from the ListBox is selected, the toolbar property is set to true.

Later in this tutorial, the Button control is used to update additional selections.

At run time, you can select which report and toolbar elements you wish to display.

You begin by adding the controls into the table at the top of the Web or Windows Form.

To add the ListBox and Button controls

1. Open the Web or Windows Form in Design view.
2. From the Toolbox, drag a Label control to the first row, column one of the table.
3. Select the Label control, and then from the Properties window, set the Text to "Select report elements to display."
4. From the Toolbox, drag a ListBox control to the first row, column two of the table.
5. Select the ListBox control, and then from the Properties window, do the following:

   Set the **ID** to "listCRVReport."

   Set the **SelectionMode** to "Multiple" (in a Windows project, "MultiExtended").
6. From the Toolbox, drag a second Label control to the first row, column three of the table.
7. Select the Label control, and then from the Properties window, set the Text to "Select toolbar elements to display."
8. From the Toolbox, drag a Button control to the third row, column one of the table.
9. Click on the Button control to select it.
10. From the Properties window:

Set the **ID** to "redisplay."

Set the **Text** to "Redisplay Report."

11. In a Windows project, resize the Button control to display the full button text.

Your next steps vary, depending on whether you are building a Web Site or a Windows project. Please choose one of the following:

Configuring the ListBox controls for a Web Site

Configuring the ListBox controls for a Windows project

Configuring the ListBox controls for a Web Site

This section explains how to configure ListBox controls for a Web Site. If you are building a Windows project, see Configuring the ListBox controls for a Windows project.

You can now create the click event handler for the Button control, and then add code to this event handler. The event handler sets various Boolean values for the toolbar properties of the CrystalReportViewer class based on the user's selections in the ListBox control.

Before creating this event handler, you need to create two enums: CeWebCRVReportOptions and CeWebCRVToolbarOptions.

These enums provide a list of selectable report elements and toolbar elements.

To create the CeWebCRVReportOptions enum

1. In Solution Explorer, right-click the Web Site name, point to Add, and then click Add New Item.
2. In the Add New Item dialog box, select Class from the Templates view.
3. In the Name field, type "CeWebCRVReportOptions", and then click Add.

   Note In Visual Studio 2005, you may be asked to place this class in an App_Code directory. Click the Yes button.
4. In the class signature, change the word class to "enum" to convert the class to an enumeration.

   Note In Visual Basic, remember to change both the opening and the closing signatures of the class to enum.
5. Because enums do not have constructors, delete the default constructor method that is provided in the C# version of the code.
6. Inside the enum, enter the values:

[Visual Basic]

```
Toolbar
Group_Tree
Main_Page
Enable_Separate_Pages
```

[end]

[C#]

```
Toolbar,
```

```
    Group_Tree,
    Main_Page,
    Enable_Separate_Pages
[end]
```

7. From the File menu, click Save All.

To create the CeWebCRVToolbarOptions enum

1. In Solution Explorer, right-click the Web Site name, point to Add, and then click Add New Item.
2. In the Add New Item dialog box, select Class from the Templates view.
3. In the Name field, type "CeWebCRVToolbarOptions", and then click Add.

   Note In Visual Studio 2005, you may be asked to place this class in an App_Code directory. Click the Yes button.
4. In the class signature, change the word class to "enum" to convert the class to an enumeration.

   Note In Visual Basic, remember to change both the opening and the closing signatures of the class to enum.
5. Because enums do not have constructors, delete the default constructor method that is provided in the C# version of the code.
6. Inside the enum, enter the values:

[Visual Basic]

```
    Group_Tree_Button
    Export_Button
    Print_Button
    View_List_Button
    Drill_Up_Button
    Page_Navigation_Button
    Go_to_Page_Button
    Search_Button
    Zoom_Button
    Crystal_Logo
[end]
```

[C#]

```
    Group_Tree_Button,
    Export_Button,
    Print_Button,
    View_List_Button,
    Drill_Up_Button,
    Page_Navigation_Button,
```

```
    Go_to_Page_Button,
    Search_Button,
    Zoom_Button,
    Crystal_Logo
```

[end]

7. From the File menu, click Save All.

You now populate the ListBox controls with the enum values, which represent the properties that are available for the CrystalReportViewer toolbar.

To populate the ListBox controls from the enums

1. Open the Web Form.
2. From the View menu, click Code.
3. Within the ConfigureCrystalReports() method, add a Not IsPostBack conditional block.

   Note You created the `ConfigureCrystalReports()` method during Appendix: Project Setup at the beginning of this tutorial. To complete this tutorial correctly, you must begin by performing Appendix: Project Setup.

[Visual Basic]

```
    If Not IsPostBack Then


    End If
```

[end]

[C#]

```
    if (!IsPostBack)
    {
    }
```

[end]

4. Within the conditional block, set the DataSource property of the listCRVReport ListBox control to the values of the CeWebCRVReportOptions enum.

[Visual Basic]

```
    listCRVReport.DataSource =
    System.Enum.GetValues(GetType(CeWebCRVReportOptions))
```

[end]

[C#]

```
    listCRVReport.DataSource =
    System.Enum.GetValues(typeof(CeWebCRVReportOptions));
```

[end]

5. Call the DataBind() method of the listCRVReport ListBox control to bind the values to the control.

[Visual Basic]

```
    listCRVReport.DataBind()
```

[end]

[C#]

```
listCRVReport.DataBind();
```

[end]

6. Next set the DataSource property of the listCRVToolbar ListBox control to the values of the CeWebCRVToolbarOptions enum.

[Visual Basic]

```
listCRVToolbar.DataSource =
System.Enum.GetValues(GetType(CeWebCRVToolbarOptions))
```

[end]

[C#]

```
listCRVToolbar.DataSource =
System.Enum.GetValues(typeof(CeWebCRVToolbarOptions));
```

[end]

7. Now call the DataBind() method of the listCRVToolbar ListBox control to bind the values to the control.

[Visual Basic]

```
listCRVToolbar.DataBind()
```

[end]

[C#]

```
listCRVToolbar.DataBind();
```

[end]

8. Outside the Not IsPostBack conditional block, bind the Chart.rpt file to the ReportSource property of the CrystalReportViewer control. For information about sample reports, see Appendix: Sample Reports' Directory.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = "C:\Program Files\Microsoft Visual
Studio 8\Crystal Reports\Samples\En\Reports\Feature Examples\Chart.rpt"
```

[end]

[C#]

```
crystalReportViewer.ReportSource = "C:\\Program Files\\Microsoft Visual
Studio 8\\Crystal Reports\\Samples\\En\\Reports\\Feature
Examples\\Chart.rpt";
```

[end]

You can now add code to the click event of the Button control. The click method must set Boolean values for the report and toolbar elements of the CrystalReportViewer class. If an element is selected, the Boolean value is set to true, and the report or toolbar element is displayed. If a property is not selected, the Boolean value is set to False, and the report or toolbar element is not displayed.

To code the redisplay Button control for a Web Site

1. Open the Web Form.
2. From the View menu, click Designer.
3. Double-click the redisplay Button control.

The code-behind class for the report appears and shows that a `redisplay_Click()` event method has been automatically generated.

4. Within the redisplay_Click() event method, call the Selected property for each item in the listCRVReport and listCRVToolbar ListBox controls.

   The Selected property returns a Boolean value to set the **CrystalReportViewer** toolbar properties.

   Note The CrystalReportViewer report and toolbar elements are set to their corresponding values in the CeWebCRVReportOptions and CeWebCRVToolbarOptions enums. The values from the enum class return a string, which you need to convert to an integer.

[Visual Basic]

```
myCrystalReportViewer.HasToggleGroupTreeButton =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Group_Tree_
Button)).Selected

myCrystalReportViewer.HasExportButton =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Export_Butt
on)).Selected

myCrystalReportViewer.HasPrintButton =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Print_Butto
n)).Selected

myCrystalReportViewer.HasViewList =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.View_List_B
utton)).Selected

myCrystalReportViewer.HasDrillUpButton =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Drill_Up_Bu
tton)).Selected

myCrystalReportViewer.HasPageNavigationButtons =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Page_Naviga
tion_Button)).Selected

myCrystalReportViewer.HasGotoPageButton =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Go_to_Page_
Button)).Selected

myCrystalReportViewer.HasSearchButton =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Search_Butt
on)).Selected

myCrystalReportViewer.HasZoomFactorList =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Zoom_Button
)).Selected

myCrystalReportViewer.HasCrystalLogo =
listCRVToolbar.Items(Convert.ToInt32(CeWebCRVToolbarOptions.Crystal_Log
o)).Selected
```

```
    myCrystalReportViewer.DisplayToolbar =
    listCRVReport.Items(Convert.ToInt32(CeWebCRVReportOptions.Toolbar)).Sel
    ected
    myCrystalReportViewer.DisplayGroupTree =
    listCRVReport.Items(Convert.ToInt32(CeWebCRVReportOptions.Group_Tree)).
    Selected
    myCrystalReportViewer.DisplayPage =
    listCRVReport.Items(Convert.ToInt32(CeWebCRVReportOptions.Main_Page)).S
    elected
    myCrystalReportViewer.SeparatePages =
    listCRVReport.Items(Convert.ToInt32(CeWebCRVReportOptions.Enable_Separa
    te_Pages)).Selected
[end]
[C#]
    crystalReportViewer.HasToggleGroupTreeButton =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Group_Tree_
    Button)].Selected;
    crystalReportViewer.HasExportButton =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Export_Butt
    on)].Selected;
    crystalReportViewer.HasPrintButton =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Print_Butto
    n)].Selected;
    crystalReportViewer.HasViewList =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.View_List_B
    utton)].Selected;
    crystalReportViewer.HasDrillUpButton =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Drill_Up_Bu
    tton)].Selected;
    crystalReportViewer.HasPageNavigationButtons =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Page_Naviga
    tion_Button)].Selected;
    crystalReportViewer.HasGotoPageButton =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Go_to_Page_
    Button)].Selected;
    crystalReportViewer.HasSearchButton =
    listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Search_Butt
    on)].Selected;
```

```
crystalReportViewer.HasZoomFactorList =
listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Zoom_Button
)].Selected;
crystalReportViewer.HasCrystalLogo =
listCRVToolbar.Items[Convert.ToInt32(CeWebCRVToolbarOptions.Crystal_Log
o)].Selected;


crystalReportViewer.DisplayToolbar =
listCRVReport.Items[Convert.ToInt32(CeWebCRVReportOptions.Toolbar)].Sel
ected;
crystalReportViewer.DisplayGroupTree =
listCRVReport.Items[Convert.ToInt32(CeWebCRVReportOptions.Group_Tree)].
Selected;
crystalReportViewer.DisplayPage =
listCRVReport.Items[Convert.ToInt32(CeWebCRVReportOptions.Main_Page)].S
elected;
crystalReportViewer.SeparatePages =
listCRVReport.Items[Convert.ToInt32(CeWebCRVReportOptions.Enable_Separa
te_Pages)].Selected;
```

[end]

You are now ready to build and run the project to customize the CrystalReportViewer toolbar.

To test the redisplay Button control

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.

   The listCRVReport and listCRVToolbar ListBox controls display a complete list of CrystalReportViewer report and toolbar options.
4. In the toolbar options listbox, select "Page_Navigation_Button", "Print_Button", and "Export_Button".
5. In the report options listbox, select "Toolbar", "Group_Tree", and "Main_Page".
6. Click Redisplay Report.

   The page reloads to display a CrystalReportViewer control with a visible toolbar, group tree, and main page. Within the toolbar, only the Page Navigation, Print, and Export buttons are visible.
7. Return to Visual Studio 2005 and click Stop to exit from debug mode.

Configuring the ListBox controls for a Windows project

This section explains how to configure ListBox controls for a Windows project. If you are building a Web Site, see Configuring the ListBox controls for a Web Site.

You can now create the click event handler for the Button control, and then add code to this event handler. The event handler sets various Boolean values for the toolbar properties of the CrystalReportViewer class based on the user's selections in the ListBox control.

Before creating this event handler, you need to create two enums: CeWinCRVReportOptions and CeWinCRVToolbarOptions.

These enums provide a list of selectable report elements and toolbar elements.

To create the CeWinCRVReportOptions enum

1. In Solution Explorer, right-click the project name that is in bold type, point to Add, and then click Class.
2. In the Add New Item dialog box, in the Name field, type "CeWinCRVReportOptions", and then click Add.

   Note In Visual Studio 2005, you may be asked to place this class in an App_Code directory. Click the Yes button.
3. In the class signature, change the word class to "enum" to convert the class to an enumeration.

   Note In Visual Basic, remember to change both the opening and the closing signatures of the class to enum.
4. Because enums do not have constructors, delete the default constructor method that is provided in the C# version of the code.
5. Inside the enum, enter the values:

[Visual Basic]

```
    Toolbar
    Group_Tree
    Status_Bar
[end]
```

[C#]

```
    Toolbar,
    Group_Tree,
    Status_Bar
[end]
```

6. From the File menu, click Save All.

To create the CeWinCRVToolbarOptions enum

1. In Solution Explorer, right-click the project name that is in bold type, point to Add, and then click Class.
2. In the Add New Item dialog box, in the Name field, type "CeWinCRVToolbarOptions", and then click Add.

Note In Visual Studio 2005, you may be asked to place this class in an App_Code directory. Click the Yes button.

3. In the class signature, change the word class to "enum" to convert the class to an enumeration.

   Note In Visual Basic, remember to change both the opening and the closing signatures of the class to enum.

4. Because enums do not have constructors, delete the default constructor method that is provided in the C# version of the code.
5. Inside the enum, enter the values:

[Visual Basic]

```
Page_Navigation_Button
Go_to_Page_Button
Close_View_Button
Print_Button
Refresh_Button
Export_Button
Group_Tree_Button
Zoom_Button
Search_Button
```

[end]

[C#]

```
Page_Navigation_Button,
Go_to_Page_Button,
Close_View_Button,
Print_Button,
Refresh_Button,
Export_Button,
Group_Tree_Button,
Zoom_Button,
Search_Button
```

[end]

1. From the **File** menu, click **Save All**.

You now populate the ListBox controls with the enum values, which represent the properties that are available for the CrystalReportViewer toolbar.

To populate the ListBox controls from the enums

1. Open the Windows Form.
2. From the View menu, click Code.
3. Within the ConfigureCrystalReports() method, set the DataSource property of the listCRVReport ListBox control to the values of the CeWinCRVReportOptions enum.

Note You created the `ConfigureCrystalReports()` method during Appendix: Project Setup at the beginning of this tutorial. To complete this tutorial correctly, you must begin by performing Appendix: Project Setup.

[Visual Basic]

```
listCRVReport.DataSource =
System.Enum.GetValues(GetType(CeWinCRVReportOptions))
```

[end]

[C#]

```
listCRVReport.DataSource =
System.Enum.GetValues(typeof(CeWinCRVReportOptions));
```

[end]

4. Set the DataSource property of the listCRVToolbar ListBox control to the values of the CeWinCRVToolbarOptions enum.

[Visual Basic]

```
listCRVToolbar.DataSource =
System.Enum.GetValues(GetType(CeWinCRVToolbarOptions))
```

[end]

[C#]

```
listCRVToolbar.DataSource =
System.Enum.GetValues(typeof(CeWinCRVToolbarOptions));
```

[end]

5. Bind the Chart.rpt file to the ReportSource property of the CrystalReportViewer control. For information about sample reports, see Appendix: Sample Reports' Directory.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = "C:\Program Files\Microsoft Visual
Studio 8\Crystal Reports\Samples\En\Reports\Feature Examples\Chart.rpt"
```

[end]

[C#]

```
crystalReportViewer.ReportSource = "C:\\Program Files\\Microsoft Visual
Studio 8\\Crystal Reports\\Samples\\Reports\\Feature
Examples\\Chart.rpt";
```

[end]

You can now add code to the click event of the Button control. The click method must set Boolean values for the report and toolbar elements of the CrystalReportViewer class. If an element is selected, the Boolean value is set to true, and the report or toolbar element is displayed. If a property is not selected, the Boolean value is set to False, and the report or toolbar element is not displayed.

To code the redisplay Button control for a Windows project

1. Open the Windows Form.
2. From the View menu, click Designer.
3. Double-click the redisplay Button control.

The code-behind class for the report appears and shows that a `redisplay_Click()` event method has been automatically generated.

4. Within the redisplay_Click() event method, call the GetSelected() method, and pass each item from the ListBox. The GetSelected() method returns a Boolean value to set the CrystalReportViewer report or toolbar properties.

   Note The CrystalReportViewer report and toolbar elements are set to their corresponding values in the CeWinCRVReportOptions and CeWinCRVToolbarOptions enums.

[Visual Basic]

```
myCrystalReportViewer.ShowPageNavigateButtons =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Page_Navigation_Butto
n)
myCrystalReportViewer.ShowGotoPageButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Go_to_Page_Button)
myCrystalReportViewer.ShowCloseButton = listCRVToolbar.
GetSelected(CeWinCRVToolbarOptions.Close_View_Button)
myCrystalReportViewer.ShowPrintButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Print_Button)
myCrystalReportViewer.ShowRefreshButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Refresh_Button)
myCrystalReportViewer.ShowExportButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Export_Button)
myCrystalReportViewer.ShowGroupTreeButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Group_Tree_Button)
myCrystalReportViewer.ShowZoomButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Zoom_Button)
myCrystalReportViewer.ShowTextSearchButton =
listCRVToolbar.GetSelected(CeWinCRVToolbarOptions.Search_Button)


myCrystalReportViewer.DisplayToolbar =
listCRVReport.GetSelected(CeWinCRVReportOptions.Toolbar)
myCrystalReportViewer.DisplayGroupTree =
listCRVReport.GetSelected(CeWinCRVReportOptions.Group_Tree)
myCrystalReportViewer.DisplayStatusBar =
listCRVReport.GetSelected(CeWinCRVReportOptions.Status_Bar)
```

[end]

[C#]

```
crystalReportViewer.ShowPageNavigateButtons =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Page_
Navigation_Button));
```

```
crystalReportViewer.ShowGotoPageButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Go_to
_Page_Button));
crystalReportViewer.ShowCloseButton = listCRVToolbar.
GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Close_View_Button));
crystalReportViewer.ShowPrintButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Print
_Button));
crystalReportViewer.ShowRefreshButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Refre
sh_Button));
crystalReportViewer.ShowExportButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Expor
t_Button));
crystalReportViewer.ShowGroupTreeButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Group
_Tree_Button));
crystalReportViewer.ShowZoomButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Zoom_
Button));
crystalReportViewer.ShowTextSearchButton =
listCRVToolbar.GetSelected(Convert.ToInt32(CeWinCRVToolbarOptions.Searc
h_Button));


crystalReportViewer.DisplayToolbar =
listCRVReport.GetSelected(Convert.ToInt32(CeWinCRVReportOptions.Toolbar
));
crystalReportViewer.DisplayGroupTree =
listCRVReport.GetSelected(Convert.ToInt32(CeWinCRVReportOptions.Group_T
ree));
crystalReportViewer.DisplayStatusBar =
listCRVReport.GetSelected(Convert.ToInt32(CeWinCRVReportOptions.Status_
Bar));
```

[end]

You are now ready to build and run the project, to customize the CrystalReportViewer toolbar.

To test the redisplay Button control

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.

3. From the Debug menu, click Start.

   The listCRVReport and listCRVToolbar ListBox controls display a complete list of CrystalReportViewer report and toolbar options.
4. In the toolbar options listbox, select "Page_Navigation_Button", "Print_Button", and "Export_Button".
5. In the report options listbox, select "Toolbar", "Group_Tree", and "Main_Page".
6. Click Redisplay Report.

   The page reloads to display a CrystalReportViewer control with a visible toolbar, group tree, and main page. Within the toolbar, only the Page Navigation, Print, and Export buttons are visible.
7. Return to Visual Studio 2005 and click Stop to exit from debug mode.

Modifying the Background Color of the Report

In this section, you learn how to modify the background color of the report.

To begin, you add a DropDownList control for background color selection.

To add controls for changing background color

1. Open the Web or Windows Form.
2. From the View menu, click Designer.
3. From the Toolbox, drag a Label control to the second row, column one of the table.
4. Select the Label control, and then from the Properties window, set the Text to "Select background color."
5. From the Toolbox, drag a DropDownList control (for Web Sites) or ComboBox control (for Windows projects) to the second row, column two of the table.
6. Select the DropDownList/ComboBox control, and then from the Properties window, set the ID/Name to "selectBackColor."

Now, you must add code to the ConfigureCrystalReports() method to set the default values for the background color list and the report components' checkboxes.

To set the controls' default values

1. Open the Web or Windows Form.
2. From the View menu, click Code.

   Next, within the `ConfigureCrystalReports()` method, you add code to set the control's default values.

   Note If you are building a Web Site, place these lines of code within the `Not IsPostBack` conditional block. If you are building a Windows project, place these lines of code in the main area of the `ConfigureCrystalReports()` method.
3. Assign the KnownColor enum to the DataSource property of the selectBackColor DropDownList.

[Visual Basic]

```
selectBackColor.DataSource = System.Enum.GetValues(GetType(KnownColor))
```

[end]

[C#]

```
selectBackColor.DataSource = System.Enum.GetValues(typeof(KnownColor));
```

[end]

4. In a Web Site, bind the data source to the selectBackColor DropDownList.

[Visual Basic]

```
selectBackColor.DataBind()
```

[end]

[C#]

```
selectBackColor.DataBind();
```

[end]

Next, you add code to the Button click event to redisplay the report based on the selectBackColor DropDownList selection.

This code varies for a Web Site compared to a Windows project. Select the appropriate procedure below for either your Web Site or Windows project.

To assign the background color selection in a Web Site

1. Open the Web Form.
2. From the View menu, click Code.
3. Above the class signature, add an "Imports" [Visual Basic] or "using" [C#] declaration to the top of the class for System.Drawing (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Drawing
```

[end]

[C#]

```
using System.Drawing;
```

[end]

4. Within the redisplay_Click() event handler, add the following code:

   From the selectBackColor DropDownList, retrieve the selected item as a string, and pass it to the `FromName()` method of the Color class. Assign the Color value to the BackColor property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.BackColor =
Color.FromName(selectBackColor.SelectedItem.Text)
```

[end]

[C#]

```
crystalReportViewer.BackColor =
Color.FromName(selectBackColor.SelectedItem.Text);
```

[end]

You are now ready to test the Redisplay Report button. Skip to that section, below.

To assign the background color selection in a Windows project

1. Open the Windows Form.
2. From the **View** menu, click **Code**.

3. Above the class signature, add an "Imports" [Visual Basic] or "using" [C#] declaration to the top of the class for System.Drawing (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Drawing
```

[end]

[C#]

```
using System.Drawing;
```

[end]

4. Within the redisplay_Click() event handler, add the following code:
5. From the **selectBackColor ComboBox**, retrieve the selected item and convert it to a KnownColor instance.

[Visual Basic]

```
Dim mySelectedKnownColor As KnownColor =
CType(selectBackColor.SelectedItem, KnownColor)
```

[end]

[C#]

```
KnownColor selectedKnownColor =
(KnownColor)selectBackColor.SelectedItem;
```

[end]

6. Create a conditional block that checks if the selected background color is not transparent.

[Visual Basic]

```
If Not mySelectedKnownColor = KnownColor.Transparent Then


End If
```

[end]

[C#]

```
if (selectedKnownColor != KnownColor.Transparent)
{
}
```

[end]

7. Within the If block, pass the KnownColor instance to the FromKnownName() method of the System.Drawing.Color class. Assign the Color value to the BackColor property of the **CrystalReportViewer** control.

[Visual Basic]

```
myCrystalReportViewer.BackColor =
System.Drawing.Color.FromKnownColor(mySelectedKnownColor)
```

[end]

[C#]

```
    crystalReportViewer.BackColor =
    System.Drawing.Color.FromKnownColor(selectedKnownColor);
[end]
```

You are now ready to test the Redisplay Report button.

To test the redisplay Button control

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.

   The DropDownList/ComboBox is displayed, along with the ListBox and Button controls that you have added in the previous procedure.
4. From the selectBackColor DropDownList, select "Blue."

   Note Remember to select the report elements, especially Main_Page, so that it will be visible!
5. Click Redisplay Report.

   The page reloads to display the report on a blue background.
6. Return to Visual Studio 2005 and click Stop to exit from debug mode.

Adding Code to Select a Report Page

In this section, you learn how to code the "Go to Page" option of the CrystalReportViewer toolbar.

The CrystalReportViewer toolbar has page navigation buttons and a textbox to select report pages. You can use the following methods from the CrystalReportViewer class to manually write code for page selections:

ShowFirstPage()

ShowLastPage()

ShowNextPage()

ShowNthPage(int PageNumber)

ShowPreviousPage()

When one of these methods is called, the selected page is displayed for the current report.

To add the TextBox and Button controls for the "Go to Page" option

1. Open the Web or Windows Form in Design view.
2. From the Toolbox, drag a TextBox control to the fourth row, column one of the table.
3. Select the TextBox control, and then from the Properties window, do the following:

   Set the **ID** (or **Name**) to "pageNumber".

   Set the **Text** property to be blank.
4. From the Toolbox, drag a Button control to the fourth row, column two of the table.
5. Select the Button control, and then from the Properties window, do the following:

   Set the **ID** (or **Name**) to "goToPage".

   Set the **Text** to "Go to Page".

To code the Click() event handler for the Button control

1. Double-click the Go to Page Button control.

   The code-behind class for the report appears and shows that a `goToPage_Click()` event handler has been automatically generated.
2. Convert the text that is typed into the TextBox control to an integer, and then pass the value to the ShowNthPage() method of the CrystalReportViewer control.

   Note You have not validated that an integer was entered into the TextBox control. For a production application, you would add a validation control configured against the TextBox control.

[Visual Basic]

```
myCrystalReportViewer.ShowNthPage(Convert.ToInt32(pageNumber.Text))
```

[end]

[C#]

```
crystalReportViewer.ShowNthPage(Convert.ToInt32(pageNumber.Text));
```

[end]

To test the goToPage button control

1. From the Build menu, select Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.

   The Chart report and all the controls that you have added are displayed.
4. Enter "3" in the pageNumber TextBox, and then click Go to Page.

   The page reloads to display page 3 of the report.
5. Return to Visual Studio 2005 and click Stop to exit from debug mode.

Modifying the Zoom Factor

By default, the CrystalReportViewer toolbar allows you to select a zoom factor from 25% to 400% by fixed increments of 25%, 50%, or 100%. In this section, you add code to allow any zoom factor that you want.

You need a TextBox control in which to type your desired zoom factor and a Button control to reload the page.

To add the TextBox and Button controls for the Zoom option

1. Open the Web or Windows Form in Design view.
2. From the Toolbox, drag a TextBox control to the fourth row, column three of the table.
3. Select the TextBox control, and then from the Properties window, do the following:

   Set the **ID** (or **Name**) to "zoomFactor"

   Set the **Text** property to be blank.
4. From the Toolbox, drag a Button control to the fourth row, column four of the table.
5. Select the Button control, and then from the Properties window, do the following:

   Set the **ID** (or **Name**) to "updateZoomFactor".

   Set the **Text** to "% Zoom".

To code the Click() event handler for the Button control

1. Double-click the updateZoomFactor Button control.

   The code-behind class for the report appears and shows that an `updateZoomFactor_Click()` event handler has been automatically generated.

2. Convert the text that is typed into the TextBox control to an integer, and then pass the value to the Zoom() method of the CrystalReportViewer control.

   Note You have not validated that an integer was entered into the TextBox control. For a production application, you would add a validation control configured against the TextBox control.

[Visual Basic]

```
myCrystalReportViewer.Zoom(Convert.ToInt32(zoomFactor.Text))
```

[end]

[C#]

```
crystalReportViewer.Zoom(Convert.ToInt32(zoomFactor.Text));
```

[end]

To test the updateZoomFactor Button control

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.

   The Chart report and all the controls that you have added are displayed.

4. Enter "38" in the zoomFactor TextBox, and then click Zoom.

   The page reloads to display the current page at 38% of its original size.

5. Return to Visual Studio 2005 and click Stop to exit from debug mode.

Searching in the Report

In this section, you learn how to search for text in a Crystal report that binds to the CrystalReportViewer control.

You need a TextBox control to type in your desired search string, a Button control to search the report, and a Label control to notify the success or failure of the search.

To add the TextBox, Button, and Label controls for the Search option

1. Open the Web or Windows Form in Design view.
2. From the Toolbox, drag a TextBox control to the fifth row, column one of the table.
3. Select the TextBox control, and then from the Properties window, do the following:

   Set the **ID** (or **Name**) to "searchText".

   Set the **Text** property to be blank.

4. From the Toolbox, drag a Button control to the fifth row, column two of the table.
5. Select the Button control, and then from the Properties window, do the following:

   Set the **ID** (or **Name**) to "search".

   Set the **Text** to "Search For Text".

6. From the Toolbox, drag a Label control to the fifth row, column three of the table.

   Set the **ID** (or **Name**) to "message".

Set the **Text** to be blank.

Set the **ForeColor** to be **Red**.

7. Create the MessageConstants class to store standard replies as string constants for the search.

   Note For instructions to create this class, see Appendix: Add a Class for Error Messages.

Next, you must call the SearchForText() method in the search Button control.

Note The SearchForText() method does not have the same behavior for a Web Site compared to a Windows project.

For a Web Site, the SearchForText() method returns True when no search results are found. For a Windows project, the SearchForText() method returns True when search results are found. Also, for a Web Site, the search finds only the first occurrence of the string; whereas, the search in a Windows project continues from where the last search result is found.

Because the method behavior is different in a Web Site compared to a Windows project, complete the corresponding step procedure below for either a Web Site or a Windows project.

To code the search_Click() event handler for a Web Site

1. Double-click the search Button control.

   The code-behind class for the report appears and shows that a `search_Click()` event handler has been automatically generated.

2. Retrieve the text that is typed into the TextBox control, and then pass the value to the SearchForText() method of the CrystalReportViewer control. Assign the method call to a Boolean variable.

   Note For the SearchDirection class to be accessible, you must include an `"Imports" [Visual Basic]` or `"using" [C#]` statement at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

[Visual Basic]

```
Dim mySearchResult As Boolean =
myCrystalReportViewer.SearchForText(searchText.Text,
SearchDirection.Forward)
```

[end]

[C#]

```
bool searchResult = crystalReportViewer.SearchForText(searchText.Text,
SearchDirection.Forward);
```

[end]

3. Create a conditional block that tests if the search is successful.

[Visual Basic]

```
If Not mySearchResult Then


Else
```

```
    End If
```
[end]
[C#]
```
    if(!searchResult)
    {
    }
    else
    {
    }
```
[end]

4. Within the If block, assign the MessageConstants.SUCCESS constant to the Text property of the message Label.

[Visual Basic]
```
    message.Text = MessageConstants.SUCCESS
```
[end]
[C#]
```
    message.Text = MessageConstants.SUCCESS;
```
[end]

5. Within the Else block, assign the MessageConstants.FAILURE constant to the Text property of the message Label.

[Visual Basic]
```
    message.Text = MessageConstants.FAILURE
```
[end]
[C#]
```
    message.Text = MessageConstants.FAILURE;
```
[end]

Skip the following procedure for a Windows project to the testing procedure that follows.

To code the search_Click() event handler for a Windows Project

1. Double-click the search Button control.

   The code-behind class for the report appears and shows that a `search_Click()` event handler has been automatically generated.

2. Retrieve the text that is typed into the TextBox control, and then pass the value to the SearchForText() method of the CrystalReportViewer control. Assign the method call to a Boolean variable.

[Visual Basic]
```
    Dim mySearchResult As Boolean =
    myCrystalReportViewer.SearchForText(searchText.Text)
```
[end]
[C#]

```
   bool searchResult = crystalReportViewer.SearchForText(searchText.Text);
```

[end]

3. Create a conditional block that tests if the search is successful.

[Visual Basic]

```
   If mySearchResult Then


   Else


   End If
```

[end]

[C#]

```
   if(searchResult)
   {
   }
   else
   {
   }
```

[end]

4. Within the If block, assign the MessageConstants.SUCCESS constant to the Text property of the message Label.

[Visual Basic]

```
   message.Text = MessageConstants.SUCCESS
```

[end]

[C#]

```
   message.Text = MessageConstants.SUCCESS;
```

[end]

5. Within the Else block, assign the MessageConstants.FAILURE constant to the Text property of the message Label.

[Visual Basic]

```
   message.Text = MessageConstants.FAILURE
```

[end]

[C#]

```
   message.Text = MessageConstants.FAILURE;
```

[end]

To test the search Button control

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.

   The Chart report and all the controls that you have added are displayed.

4. Enter "China" in the searchText TextBox, and then click Search.

   The page reloads to highlight the search result and to display a success message.

5. Enter "hello" in the searchText TextBox, and then click Search.

   The page reloads to display a failure message.

6. Return to Visual Studio 2005 and click Stop to exit from debug mode.

The remaining customization options are only available for the Web version of the CrystalReportViewer control.

Therefore, if you are developing a Web Site, continue to Adding a Border to the Report for a Web Site.

Otherwise, if you are developing a Windows project, continue to Conclusion.

Adding a Border to the Report for a Web Site

In this section, you learn how to add a customized border to the Crystal report for a Web Site. For a Windows project, the properties for border width, style, and color are not available.

The border styles are listed in the BorderStyle enumeration. The colors are listed in the KnownColor enumeration. However, the BorderColor property of the CrystalReportViewer control takes in values from the Color class. Therefore, you have to convert the KnownColor value to the Color value.

To begin, you add the necessary controls to the Web Form. You need a TextBox control, two DropDownList controls, and a Button control to draw the border.

To add the controls to set border width, style, and color

1. Open the Web Form.
2. From the View menu, click Designer.
3. From the Toolbox, drag a Label control to the sixth row, column one of the table.
4. Select the Label control, and then from the Properties window, set the Text to "Border Width."
5. From the Toolbox, drag a TextBox control into the same table cell as the Label control.
6. Select the TextBox control, and then from the Properties window, set the ID (or Name) to "borderWidth".
7. From the Toolbox, drag a second Label control to the sixth row, column two of the table.
8. Select the Label control, and then from the Properties window, set the Text to "Border Style."
9. From the Toolbox, drag a DropDownList control into the same table cell as the Label control.
10. Select the DropDownList control, and then from the Properties window, set the ID (or Name) to "selectBorderStyle".
11. From the Toolbox, drag a third Label control to the sixth row, column three of the table.
12. Select the Label control, and then from the Properties window, set the Text to "Border Color."

13. From the Toolbox, drag a DropDownList control into the same table cell as the Label control.
14. Select the DropDownList control, and then from the Properties window, set the ID (or Name) to "selectBorderColor".
15. From the Toolbox, drag a Button control to the sixth row, column four of the table.
16. Select the Button control, and then from the Properties window, do the following:

    Set the **ID** (or **Name**) to "drawBorder".

    Set the **Text** property to "Draw Border".

You now must populate the DropDownList controls with the border styles or border colors that are available for the CrystalReportViewer control. The DropDownList controls are populated in the ConfigureCrystalReports() method.

For a Web Site, Border Styles are stored in the System.Web.UI.WebControls.BorderStyle enumeration. Border Colors are retrieved from the System.Drawing.KnownColor enumeration.

To populate the DropDownList controls

1. Above the class signature, add an "Imports" [Visual Basic] or "using" [C#] declaration to the top of the class for System.Web.UI.WebControls and System.Drawing (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Web.UI.WebControls
Imports System.Drawing
```

[end]

[C#]

```
using System.Web.UI.WebControls;
using System.Drawing;
```

[end]

2. Within the ConfigureCrystalReports() method, in the Not IsPostBack conditional block, assign the BorderStyle enum to the DataSource property of the selectBorderStyle DropDownList.

[Visual Basic]

```
selectBorderStyle.DataSource =
System.Enum.GetValues(GetType(BorderStyle))
```

[end]

[C#]

```
selectBorderStyle.DataSource =
System.Enum.GetValues(typeof(BorderStyle));
```

[end]

3. Bind the data source to the selectBorderStyle DropDownList.

[Visual Basic]

```
selectBorderStyle.DataBind()
```

[end]

[C#]

```
    selectBorderStyle.DataBind();
```
[end]

4. Still within the Not IsPostBack conditional block, assign the KnownColor enum to the DataSource property of the selectBorderColor DropDownList.

[Visual Basic]
```
    selectBorderColor.DataSource =
    System.Enum.GetValues(GetType(KnownColor))
```
[end]
[C#]
```
    selectBorderColor.DataSource =
    System.Enum.GetValues(typeof(KnownColor));
```
[end]

5. Bind the data source to the selectBorderColor DropDownList.

[Visual Basic]
```
    selectBorderColor.DataBind()
```
[end]
[C#]
```
    selectBorderColor.DataBind();
```
[end]

Next, you assign values to the BorderWidth, BorderStyle, and BorderColor properties of the CrystalReportViewer control.

To code the Draw Border Button control

1. Open the Web Form.
2. From the View menu, click Designer.
3. Double-click the Draw Border button control.

   The code-behind class for the report appears and shows that a `drawBorder_Click()` event handler has been automatically generated.
4. Within the drawBorder_Click() event handler, assign the text that is typed into the borderWidth TextBox to the BorderWidth property of the CrystalReportViewer control.

   Note You have not validated that an integer was entered into the TextBox control. For a production application, you would add a validation control configured against the TextBox control.

[Visual Basic]
```
    myCrystalReportViewer.BorderWidth =
    Unit.Parse(borderWidth.Text.ToString())
```
[end]
[C#]
```
    crystalReportViewer.BorderWidth = Convert.ToInt32(borderWidth.Text);
```
[end]

5. From the selectBorderStyle DropDownList, retrieve the selected index, and convert it into a BorderStyle value. Assign the BorderStyle value to the BorderStyle property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.BorderStyle =
CType(selectBorderStyle.SelectedIndex, BorderStyle)
```

[end]

[C#]

```
crystalReportViewer.BorderStyle =
(BorderStyle)selectBorderStyle.SelectedIndex;
```

[end]

6. From the selectBorderColor DropDownList, retrieve the selected item as a string, and pass it to the FromName() method of the Color class. Assign the Color value to the BorderColor property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.BorderColor =
Color.FromName(selectBorderColor.SelectedItem.Text)
```

[end]

[C#]

```
crystalReportViewer.BorderColor =
Color.FromName(selectBorderColor.SelectedItem.Text);
```

[end]

To draw a border around the Crystal report

1. From the Build menu, click Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.
4. For the border width, type "10".
5. For the border style, select "Double".
6. For the border color, select "SteelBlue".
7. Click the Draw Border button.

   The page reloads to display a border around the Crystal report.
8. Return to Visual Studio 2005 and click Stop to exit from debug mode.

Configuring Session Persistence for a Web Site

In this section, you learn how to configure Session persistence for button click events.

When a Web page is reloaded on a button click event, the changes that are made to the CrystalReportViewer object model are lost.

To demonstrate lack of persistence for a Web Site

1. From the Build menu, select Build Solution.
2. If you have any build errors, go ahead and fix them now.
3. From the Debug menu, click Start.
4. From the selectBackColor DropDownList, select "Blue."
5. Click Redisplay Report.

The page reloads to display the report with no toolbar, on a blue background.

6. For the border width, type "10".
7. For the border style, select "Double".
8. For the border color, select "SteelBlue".
9. Click Draw Border.

   The page reloads to display a border around the Crystal report, and the background color is no longer blue.
10. Enter "3" in the pageNumber TextBox, and then click Go to Page.

    The page reloads to display page 3 of the report, and the border around the report is no longer visible.
11. Return to Visual Studio 2005 and click Stop to exit from debug mode.

You must add persistence code to your application, so that changes that are made within the CrystalReportViewer object model are persisted when Web pages are reloaded.

To begin, you add persistence code for border values to the `drawBorder_Click()` event handler, where those values are first assigned. Then, in the `ConfigureCrystalReports()` method, the values that are stored in Session are retrieved and assigned to the respective property of the CrystalReportViewer class.

To add Session assignment code to the drawBorder_Click() event handler

In the drawBorder_Click() event handler, following the existing code add four Session assignments for BackColor, BorderColor, BorderStyle, and BorderWidth.

[Visual Basic]

```
Session("myBorderColor") = myCrystalReportViewer.BorderColor.ToString()
Session("myBorderStyle") = myCrystalReportViewer.BorderStyle
Session("myBorderWidth") = myCrystalReportViewer.BorderWidth
```

[end]

[C#]

```
Session["borderColor"] = crystalReportViewer.BorderColor.ToString();
Session["borderStyle"] = crystalReportViewer.BorderStyle;
Session["borderWidth"] = crystalReportViewer.BorderWidth;
```

[end]

You are now ready to retrieve these values out of Session in the `ConfigureCrystalReports()` method.

To add Session retrieval code to the ConfigureCrystalReports() method

1. At the bottom of the ConfigureCrystalReports() method, create an If block that checks whether the BackColor Session variable is not null. If it is not, then within the If block, retrieve the BackColor property from Session and cast it to a string. Pass the string to the FromName() method of the Color class and assign the Color instance to the BackColor property of the CrystalReportViewer instance.

[Visual Basic]

```
myCrystalReportViewer.BackColor =
Color.FromName(CType(Session("myBackColor"), String))
```

[end]

[C#]

```
if (Session["backColor"] != null)
{
    crystalReportViewer.BackColor =
Color.FromName((string)Session["backColor"]);
}
```

[end]

2. Create a second If block that checks whether the BorderColor Session variable is not null. If it is not, then within the If block, retrieve the BorderColor property from Session and cast it to a string. Pass the string to the FromName() method of the Color class and assign the Color instance to the BorderColor property of the CrystalReportViewer instance.

[Visual Basic]

```
myCrystalReportViewer.BorderColor =
Color.FromName(CType(Session("myBorderColor"), String))
```

[end]

[C#]

```
if (Session["borderColor"] != null)
{
    crystalReportViewer.BorderColor =
Color.FromName((string)Session["borderColor"]);
}
```

[end]

3. Create a third If block that checks whether the BorderStyle Session variable is not null. If it is not, then within the If block, retrieve the BorderStyle property from Session and cast it to BorderStyle.

[Visual Basic]

```
myCrystalReportViewer.BorderStyle = CType(Session("myBorderStyle"),
BorderStyle)
```

[end]

[C#]

```
if (Session["borderStyle"] != null)
{
    crystalReportViewer.BorderStyle =
(BorderStyle)Session["borderStyle"];
}
```

[end]

4. Create a fourth If block that checks whether the BorderWidth Session variable is not null. If it is not, then within the If block, retrieve the BorderWidth property from Session and convert it to an integer.

[Visual Basic]

```
    myCrystalReportViewer.BorderWidth =
    Convert.ToInt32(Session("myBorderWidth"))
```

[end]

[C#]

```
    if (Session["borderStyle"] != null)
    {
        crystalReportViewer.BorderWidth =
    Convert.ToInt32(Session["borderStyle"]);
    }
```

[end]

In the next procedure, you add Session persistence to the redisplay Button click event handler.

To code the redisplay Button control

Within the `redisplay_Click()` event handler, assign the selected item from the **selectBackColor DropDownList** to Session.

[Visual Basic]

```
    Session("myBackColor") = selectBackColor.SelectedItem.Text
```

[end]

[C#]

```
    Session["backColor"] = selectBackColor.SelectedItem.Text;
```

[end]

To code the drawBorder Button control

1. Within the drawBorder_Click() event handler, assign the Text property of the borderWidth TextBox to Session.

[Visual Basic]

```
    Session("myBorderWidth") = borderWidth.Text
```

[end]

[C#]

```
    Session["borderWidth"] = borderWidth.Text;
```

[end]

2. Assign the selected index from the selectBorderStyle DropDownList to Session.

[Visual Basic]

```
    Session("myBorderStyle") = selectBorderStyle.SelectedIndex
```

[end]

[C#]

```
    Session["borderStyle"] = selectBorderStyle.SelectedIndex;
```

[end]

3. Assign the selected item from the selectBorderColor DropDownList to Session.

[Visual Basic]

```
    Session("myBorderColor") = selectBorderColor.SelectedItem.Text
```

[end]

```
[C#]
    Session["borderColor"] = selectBorderColor.SelectedItem.Text;
[end]
```

You are now ready to build and run the project, to verify that the changes made to the report are persisted through button click events.

To test the project, you can follow the instructions listed in the procedures at the top of this section.

To learn about modifying the visual appearance of the CrystalReportViewer toolbar, continue to the next section.

Modifying Graphics and Cascading Style Sheets for a Web Site

The visual appearance of the CrystalReportViewer control is created through the use of graphics and a cascading style sheet (CSS).

CrystalReportViewer toolbar graphics

The CrystalReportViewer toolbar consist of different graphics that represent most of the properties available. The graphics are stored in the viewer's virtual directory for your version of Crystal Reports for Visual Studio 2005. For more information, see Appendix: Viewers' Virtual Directory.

For a Web Site, the toolbar uses graphics to represent the following properties:

- Group tree
- Export
- Print
- Drill up
- Page navigation: first, last, previous and next page
- Go to page
- Search
- Business Objects logo

To modify the CrystalReportViewer toolbar graphics

1. Navigate to the viewer's virtual directory that is specified in Appendix: Viewers' Virtual Directory.
2. Double-click the "images" folder.
3. Double-click the "toolbar" folder.

   Within the "toolbar" folder, you can find all the images that are used for the CrystalReportViewer toolbar.
4. Replace the images with other images of your choice, or open the images in a graphics editor to modify the images. However, the image name must remain the same.

CrystalReportViewer cascading style sheet

The CrystalReportViewer control uses a default cascading style sheet to customize its appearance.

Conclusion

You have successfully added code to customize the report that binds to the CrystalReportViewer control. You have also learned how to select a page, change the zoom factor, search for text, choose a print mode, and customize the border.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_CRVObjMod_CustomizeViewer

C# Windows project: CS_Win_CRVObjMod_CustomizeViewer

Visual Basic Web Site: VB_Web_CRVObjMod_CustomizeViewer

Visual Basic Windows project: VB_Win_CRVObjMod_CustomizeViewer

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorials

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Persisting the ReportDocument Object Model Using Session

Persisting the ReportDocument Object Model Using Session

Introduction

In this tutorial, you use the ReportDocument object model and make programmatic changes to a report at runtime. You also learn how to use Session to persist those changes across Web page reloads.

Because only Web applications require Session persistence, this tutorial does not apply to Windows projects.

> **Note** For an overview of persistence, see Appendix: Which Persistence Approach Should I Use with Crystal Reports? under SDK Fundamentals.

When using Session to persist a ReportDocument object, are any other persistence models involved?

Yes. In an ASP.NET Web application, objects typically use Session for persistence, while Web server controls use ViewState for persistence. Because a Crystal Reports for Visual Studio Web application uses objects and Web server controls to interact with reports, persistence is shared by Session and ViewState:

- Session persists the ReportDocument object model, which interacts with the report programmatically at runtime.
- ViewState persists the CrystalReportViewer control, which displays the report. In particular, ViewState persists the display properties that are set in the toolbar of the CrystalReportViewer. ViewState also persists any events (such as zoom, NextPage) that are triggered from buttons in the toolbar.

ViewState persistence is managed automatically. Therefore, in this tutorial you only code Session persistence.

Overview of this Tutorial

The project that you create in this tutorial uses a Web Form that contains a CrystalReportViewer control and two buttons that change the sort order of the report within the ReportDocument object model.

First, you construct the project without Session persistence. Without persistence, you can see that the change in sort order lasts no longer than a button click event.

Then, you add Session persistence to the project. The ReportDocument instance is placed into Session at the time of creation, and at any point where the state of the ReportDocument instance is changed.

Whenever the report needs to be redisplayed, the ReportDocument instance is pulled out of Session and bound to the CrystalReportViewer control. This guarantees that every time the report is displayed, the user views the most up-to-date version of the ReportDocument instance.

> **Note** In this tutorial you use Session, because you will persist modifications to the ReportDocument object model. If you plan to use only the limited object model contained within the CrystalReportViewer control, use ViewState exclusively.

Configuring Buttons on the Web Form

To begin, you add two buttons to the Web Form, name them, and then create click events for each one.

To add buttons to the Web Form

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. Open the **Default.aspx** page.
2. From the **View** menu, click **Designer**.
3. Click the **CrystalReportViewer** control and press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press the ENTER key.

   The CrystalReportViewer control drops by one line.
4. From the **Toolbox**, drag a **Button** Web server control above the **CrystalReportViewer** control.
5. Click to the right of the **Button** control, so that a flashing cursor appears, and then press the spacebar twice.
6. From the **Toolbox**, drag a second **Button** Web server control to the right of the first **Button** control.

To set the Text and ID properties for each Button control

1. Click on the first Button Web server control to select it.
2. From the **Properties** window:

   Set the **ID** to "sortOrderDescending."

   Set the **Text** to "Change Sort Order to Descending."
3. Click on the second Button Web server control to select it.
4. From the **Properties** window:

   Set the **ID** to "sortOrderAscending."

   Set the **Text** to "Change Sort Order to Ascending."

Creating click events for each Button control

1. Double-click the first Button Web server control.

   The code-behind class opens and a new event method, `sortOrderDescending_Click()`, is created at the bottom of the class.
2. Return to the design view of the **Default.aspx** page.
3. Double-click the second Button Web server control.

   The code-behind class opens and a new event method, `sortOrderAscending_Click()`, is created at the bottom of the class.

You are now ready to program the ReportDocument object model of the report within the event method, to set the sort order to ascending or descending.

Programming the Button Events with Sort Orders

In this procedure you program the ReportDocument object model of the report within the event method, and set the sort order to ascending in one event method and to descending in the other event method.

To program the ReportDocument object model of the report in the sortOrderDescending_Click event method

1. Open the code-behind class and locate the `sortOrderDescending_Click()` event method.
2. Within the event method, retrieve the SortFields property from the DataDefinition property of the report instance and assign it to an instance of the SortFields indexed class.

[Visual Basic]

```
Dim mySortFields As SortFields =
hierarchicalGroupingReport.DataDefinition.SortFields
```

[end]

[C#]

```
SortFields sortFields =
hierarchicalGroupingReport.DataDefinition.SortFields;
```

[end]

3. Retrieve the first SortField instance from the SortFields indexed class and assign it to a variable named firstSortField.

   Note The SortFields indexed class is 0-based.

[Visual Basic]

```
Dim firstSortField As SortField = mySortFields(0)
```

[end]

[C#]

```
SortField firstSortField = sortFields[0];
```

[end]

4. Set the SortDirection property of firstSortField to descending order, by use of the DescendingOrder selection of the SortDirection enum.

[Visual Basic]

```
firstSortField.SortDirection = SortDirection.DescendingOrder
```

[end]

[C#]

```
firstSortField.SortDirection = SortDirection.DescendingOrder;
```

[end]

5. Now re-assign this report (with its sort that has been modified in the ReportDocument object model) to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = hierarchicalGroupingReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = hierarchicalGroupingReport;
```

[end]

In the next procedure, you enter the code for the `sortOrderAscending_Click()` event method. The code is identical, except for the SortDirection property setting.

To program the ReportDocument object model of the report in the sortOrderAscending_Click event method

1. Open the code-behind class and locate the `sortOrderAscending_Click()` event method.
2. Within the event method, retrieve the SortFields property from the DataDefinition property of the report instance and assign it to an instance of the SortFields indexed class.

[Visual Basic]

```
Dim mySortFields As SortFields =
hierarchicalGroupingReport.DataDefinition.SortFields
```

[end]

[C#]

```
SortFields sortFields =
hierarchicalGroupingReport.DataDefinition.SortFields;
```

[end]

3. Retrieve the first SortField instance from the SortFields indexed class and assign it to a variable named firstSortField.

   Note The SortFields indexed class is 0-based.

[Visual Basic]

```
Dim firstSortField As SortField = mySortFields(0)
```

[end]

[C#]

```
SortField firstSortField = sortFields[0];
```

[end]

4. Set the SortDirection property of the firstSortField instance to ascending order, by use of the AscendingOrder selection of the SortDirection enum.

[Visual Basic]

```
firstSortField.SortDirection = SortDirection.AscendingOrder
```

[end]

[C#]

```
firstSortField.SortDirection = SortDirection.AscendingOrder;
```

[end]

5. Now re-assign this report (with its sort that has been modified in the ReportDocument object model) to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = hierarchicalGroupingReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = hierarchicalGroupingReport;
```

[end]

This completes the programming of the ReportDocument object model of the report within each event method. In the next section, you test whether these sort changes work, and whether they are persisted when Session has not been applied.

Testing and Determining Persistence Failures

In this section, you see that, without the use of Session for persistence, the sort changes are lost when the page is reloaded to reflect changes to display settings.

To test the programmatic changes to sort order coded in the project

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, go fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time that you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   If no build errors appear, the Default.aspx page loads into your browser, with the Hierarchical Grouping report generated on the form.
4. Observe the sort order carefully. You may want to print the page to compare the changes in sort order.
5. Click the **Change Sort Order to Descending** button.

   The sort order reverses.
6. Click the **Change Sort Order to Ascending** button.

   The sort order returns to its original (ascending) order.

In the next procedure, you verify whether the sort order is persisted when other changes to the page (such as display settings) are applied.

To test whether the sort changes are persisted when display settings are altered

1. Click the **Change Sort Order to Descending** button.

   The sort order is reversed.
2. From the toolbar of the report, adjust the zoom from 100% to 125%.

   The page reloads the report at 125%, but the sort order reversal has not been persisted.
3. Once again, click the **Change Sort Order to Descending** button.

   The sort order is reversed.
4. From the toolbar of the report, adjust the zoom from 125% to 100%.
5. The page reloads the report at 100%, but the sort order reversal has not been persisted.
6. Return to Visual Studio and click **Stop** to exit from debug mode.

From this test you can see that, at this point in the tutorial, persistence fails. The sort order is only applied when one of the sort buttons are clicked. The sort order is discarded when the user interacts with the page to adjust the zoom or open another page in a multipage report. Therefore, you must add code to persist the sort order change that is made within the ReportDocument object model with Session.

Adding Session Code

In this section, you learn how to use Session to persist the change to the sort order of the report. To the `ConfigureCrystalReports()` method that you created when you set up your project (see Appendix: Project Setup), you add code that checks for Session and immediately reloads the report instance into Session when a user adjusts the sort order.

To add Session checking code to the ConfigureCrystalReports() method

1. Within the `ConfigureCrystalReports()` method, before the existing code, create an if/else conditional block that checks whether a Session object named hierarchicalGroupingReport exists.

   You can use an identifier name of your choice for this Session object. Use a unique identifier, such as the name of the report instance.

[Visual Basic]

```
If (Session("hierarchicalGroupingReport") Is Nothing) Then


Else


End If
```

[end]

[C#]

```
if(Session["hierarchicalGroupingReport"] == null)
{
}
else
{
}
```

[end]

2. If you are using an embedded report, move the line of code that declares and instantiates the report from its present location up into the If block.

[Visual Basic]

```
If (Session("hierarchicalGroupingReport") Is Nothing) Then
    hierarchicalGroupingReport = New Hierarchical_Grouping()
Else


End If
```

[end]

[C#]

```
if(Session["hierarchicalGroupingReport"] == null)
{
   hierarchicalGroupingReport = new Hierarchical_Grouping();
}
else
{
}
```

[end]

3. If you are using a non-embedded report, move the two lines of code that declare and instantiate the report, and that load the report from the directory path up from their present location into the If block.

[Visual Basic]

```
If (Session("hierarchicalGroupingReport") Is Nothing) Then
    hierarchicalGroupingReport = New ReportDocument()
    hierarchicalGroupingReport.Load("C:\Program Files\Microsoft Visual
Studio 2005\Crystal Reports\Samples\Reports\Feature
Examples\Hierarchical Grouping.rpt")
Else

End If
```

[end]

[C#]

```
if(Session["hierarchicalGroupingReport"] == null)
{
   hierarchicalGroupingReport = new ReportDocument();
   hierarchicalGroupingReport.Load("C:\Program Files\Microsoft Visual
Studio 2005\Crystal Reports\Samples\Reports\Feature
Examples\Hierarchical Grouping.rpt");
}
else
{
}
```

[end]

4. Within the If block, assign the report into Session, using the report variable name as the Session identifier string.

[Visual Basic]

```
    Session("hierarchicalGroupingReport") = hierarchicalGroupingReport
[end]
```

[C#]

```
    Session["hierarchicalGroupingReport"] = hierarchicalGroupingReport;
[end]
```

This completes the If block. The If block is performed if the report does not exist in Session. The Else block is performed if the report is found in Session; therefore, the purpose of the Else block is to retrieve the report from Session into a report instance.

5. Within the Else block, assign the report that is stored in Session to the report instance.

[Visual Basic]

```
    hierarchicalGroupingReport =
    CType(Session("hierarchicalGroupingReport"), ReportDocument)
[end]
```

[C#]

```
    hierarchicalGroupingReport =
    (ReportDocument)Session["hierarchicalGroupingReport"];
[end]
```

Note Because Session only returns generic objects, you must cast the report to a report type. Whether you use embedded or non-embedded reports, cast the retrieved object to the ReportDocument type.

Outside and below the conditional block, the remaining code binds the report instance to the ReportSource property of the CrystalReportViewer control.

As you may remember from Project Setup (see Appendix: Project Setup), the `ConfigureCrystalReports()` method is called each time a page is reloaded. Therefore, your modification of this method guarantees that the most current report (either new, or stored in Session) is always retrieved and loaded. However, nothing has yet been written to actually update the report instance. In the next section, you update the report instance in Session each time that the sort order is modified.

To update the report instance in Session for the sortOrderDescending_Click event method

1. In the `sortOrderDescending_Click()` event method, delete the last line of code which bound the CrystalReportViewer control to the report.

   This should leave you with the three lines of code that change the sort order direction.

2. Below the three lines of code that change the sort direction, assign the hierarchicalGroupingReport instance (that has been updated with a new sort order direction) into Session using the same identifier as was used in the `ConfigureCrystalReports()` method.

   Note For simplicity, copy/paste this line of code from the `ConfigureCrystalReports()` method.

[Visual Basic]

```
    Session("hierarchicalGroupingReport") = hierarchicalGroupingReport
[end]
```

[C#]

```
Session["hierarchicalGroupingReport"] = hierarchicalGroupingReport;
```

[end]

3. Finally, add a call to the `ConfigureCrystalReports()` method, which gets the latest hierarchicalGroupingReport instance from Session (the one you have updated a moment ago), and then bind it to the CrystalReportViewer control.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

You now want to repeat the same steps for the second event method.

To update the report instance in Session for the sortOrderAscending_Click event method

1. In the `sortOrderAscending_Click()` event method, delete the last line of code which bound the CrystalReportViewer control to the report.

   This should leave you with the three lines of code that change the sort order direction.

2. Below the three lines of code that change the sort direction, assign the hierarchicalGroupingReport instance (that has been updated with a new sort order direction) into Session using the same identifier as was used in the `ConfigureCrystalReports()` method.

   For simplicity, copy and paste this line of code from the `ConfigureCrystalReports()` method.

[Visual Basic]

```
Session("hierarchicalGroupingReport") = hierarchicalGroupingReport
```

[end]

[C#]

```
Session["hierarchicalGroupingReport"] = hierarchicalGroupingReport;
```

[end]

3. Finally, add a call to the `ConfigureCrystalReports()` method, which will get the latest hierarchicalGroupingReport instance from Session (the one you updated a moment ago) and then bind it to the CrystalReportViewer control.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

You are now ready to test whether the sort changes are being persisted successfully.

Testing and Determining Persistence Success

In this section, you test whether the sort changes are being persisted successfully now that Session is used for persistence.

To test whether sort changes are being persisted successfully

1. From the **Build** menu, click **Build Solution**.

   If you have any build errors, fix them now.

2. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   If no build errors appear, the Default.aspx page loads into your browser, with the Hierarchical Grouping report generated on the form.

3. Observe the sort order carefully. You may want to print the page to compare the changes in sort order.

4. Click the **Change Sort Order to Descending** button.

   The sort order reverses.

5. From the toolbar of the report, adjust the zoom from 100% to 125%.

   The page reloads the report at 125%. The sort order reversal has been persisted.

6. Click the **Change Sort Order to Ascending** button.

   The sort order is restored.

7. From the toolbar of the report, adjust the zoom from 125% to 100%.

   The page reloads the report at 100%. The sort order retains its setting from the previous step.

From this test you can see that persistence now succeeds.

Conclusion

All of the tutorials in this section, ReportDocument Object Model Tutorials, perform operations on classes, properties and methods of the ReportDocument object model.

Those operations are not persisted unless they are stored in Session, as this tutorial demonstrates.

Even if your intention were only to display the report once, the user may choose to alter the display properties by going to another page or adjusting the zoom level, and these would result in page reloads that lose the operations you have performed against the ReportDocument object model.

Therefore, always remember to place your embedded reports (or external reports loaded into an instance of ReportDocument) into Session. Then, after each modification you perform against that ReportDocument instance, replace that instance in the Session variable to ensure the changes will be persisted on every page reload.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_Session

Visual Basic Web Site: VB_Web_RDObjMod_Session

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Logging onto a Secure SQL Server Database Using SQL Authentication

Logging onto a Secure SQL Server Database Using SQL Authentication

Introduction

In this tutorial, you learn how to add logon code to display a report that contains information from a secure SQL Server database.

To log onto a secure SQL Server database, you use classes from the ReportDocument object model.

The ReportDocument object has a Database property that returns a Database instance. This Database instance contains the database information for the report, including a Tables property that returns a Tables indexed class instance. Individual Table instances can then be retrieved from the Tables indexed class.

Logon occurs at the granular level of each Table instance, which must be granted individual access to the secure SQL Server. This is done by placing logon information into a ConnectionInfo instance, and then, within a for loop, applying that ConnectionInfo instance to the ConnectionInfo Property of each Table instance.

The properties of the ConnectionInfo class include the following:

ServerName

DatabaseName

UserID

Password

IntegratedSecurity (not used in this tutorial)

Note If you wish to work through a tutorial that uses Windows authentication (and therefore uses the IntegratedSecurity property rather than the UserID and Password properties), see Logging onto a Secure SQL Server Database Using Integrated Security.

If you choose only to set the DatabaseName, UserID, and Password properties, you will be logged onto the default server and database specified within the report. However, if you choose to assign an alternate ServerName property, you can redirect the report to a different server at runtime.

You begin by creating a report containing data from a secure SQL server database.

You can complete this tutorial by using classes of the CrystalReportViewer object model; however, the ReportDocument object model is recommended.

To build this tutorial using the CrystalReportViewer object model, see Logging onto a Secure SQL Server Database in the CrystalReportViewer tutorials.

Creating a Report Connected to a Secure SQL Server Database

To begin, create a report that draws its information from the Northwind database.

Note Northwind is a sample database provided with SQL Server.

Some setup is required as a prerequisite to this tutorial.

Prerequisite Database Setup

1. SQL Server configuration:

   If you have SQL Server (or the OEM version, MSDE) installed, it must be configured to require SQL Server Authentication to work with this tutorial.

   If you do not have SQL Server (or the OEM version, MSDE) installed, you must install MSDE with SQL Server Authentication set to "True" to work with this tutorial.
2. The Northwind database provided with SQL Server must be installed and verified that it accepts SQL Server Authentication.
3. A limited access account must be created for use within the Web site.

To install MSDE with SQL Server Authentication, or the Northwind database, go to the following sections from Appendix: System Setup in this documentation:

Appendix: MSDE Installation with Windows or SQL Server Authentication

Appendix: Northwind Database Installation

Appendix: Security: Creating a Limited Access Database Account

Once you have configured SQL Server and the Northwind database according to the sections above, you are ready to create a report that draws its information securely from a Northwind database.

To create a report with secure data from the Northwind database

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the **Crystal Report** template.
3. In the **Name** field, enter the name "NorthwindCustomers.rpt" and click **Open**.

   Note If you have not registered before, you may be asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
4. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
5. In the **Choose an Expert** panel, select **Standard**. Click **OK**.

   The Standard Report Creation Wizard window appears.
6. In the **Available Data Sources** panel, expand the **Create New Connection** folder.
7. From the subfolder that opens, expand the **OLE DB (ADO)** folder.

   The OLE DB (ADO) window appears.
8. Select **Microsoft OLE DB Provider for SQL Server** and click **Next**.
9. Enter the values for your database server, user id and password into the **Server**, **User ID** and **Password** fields.
10. From the **Database** drop down list, select "Northwind."

Leave the Integrated Security checkbox unchecked because you are using SQL Server authentication instead of NT authentication.

11. Click **Finish**.

    The OLE DB folder is now expanded, showing your database server and within it, the Northwind database.

12. Expand the nodes **Northwind**, **dbo**, and **Tables**, and then select the **Customers** table.
13. Click the **>** symbol to move the table into the **Selected Tables** panel, then click **Next**.
14. Hold down the Ctrl key while clicking **CompanyName**, **ContactName** and **City**.
15. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click **Next**.
16. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, then click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

    The NorthwindCustomers report is created and loaded into the main window of Visual Studio.

You are now ready to bind the report to the CrystalReportViewer control and set the database logon programmatically.

Binding the Report

In Appendix: Project Setup, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous step, you have added a NorthwindCustomers report to the project.

In this section, you instantiate the NorthwindCustomers report and bind it to the CrystalReportViewer control. Then you test whether the report displays correctly when current values have not been set for its parameter field.

You can instantiate and bind the report in two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the report as a non-embedded report.

To instantiate the NorthwindCustomers report as an embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the NorthwindCustomers report wrapper class, using the variable name northwindCustomersReport. Set its access modifier to private.

[Visual Basic]

```
Private northwindCustomersReport As NorthwindCustomers
```

[end]

[C#]

```
private NorthwindCustomers northwindCustomersReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
northwindCustomersReport = New NorthwindCustomers()
```

[end]

[C#]

```
northwindCustomersReport = new NorthwindCustomers();
```

[end]

5. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer control to the instantiated report class (variable name: northwindCustomersReport).

[Visual Basic]

```
myCrystalReportViewer.ReportSource = northwindCustomersReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = northwindCustomersReport;
```

[end]

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to log onto the database.

To instantiate the NorthwindCustomers report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, using the variable name northwindCustomersReport. Set its access modifier to private.

[Visual Basic]

```
Private northwindCustomersReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument northwindCustomersReport;
```

[end]

Note The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
northwindCustomersReport = New ReportDocument()
```

[end]

[C#]

```
northwindCustomersReport = new ReportDocument();
```

[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("NorthwindCustomers.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("NorthwindCustomers.rpt");
```

[end]

   For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

   Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"NorthwindCustomers.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"NorthwindCustomers.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
northwindCustomersReport.Load(reportPath)
```

[end]

[C#]

```
northwindCustomersReport.Load(reportPath);
```

[end]

7. Bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = northwindCustomersReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = northwindCustomersReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: northwindCustomersReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to log onto the database.

To test the loading of the NorthwindCustomers report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The NorthwindCustomers report does not display, because the database logon code has not been added.

   Note Results may vary, depending on the version of Crystal Reports that you use. For example, if you have Crystal Reports 10 or higher installed, a form appears and requests that you provide database logon information for that report. This is a new feature of Crystal Reports Developer. If you are running a previous version of Crystal Reports, an exception is thrown. In either case, you must follow the next step procedure to create a fully functional application.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Adding the Report Logon Code

You are now ready to add the logon code to the code-behind class. You begin by creating a private helper method, `SetDBLogonForReport()`.

To create and code the SetDBLogonForReport() method

1. Return to the code-behind class for this Web or Windows Form.
2. At the bottom of the class, create a new private method named `SetDBLogonForReport()` with two parameters, ConnectionInfo and ReportDocument.

[Visual Basic]

```
Private Sub SetDBLogonForReport(ByVal myConnectionInfo As ConnectionInfo,
ByVal myReportDocument As ReportDocument)


End Sub
```

[end]

[C#]

```
private void SetDBLogonForReport(ConnectionInfo connectionInfo,
ReportDocument reportDocument)
{
}
```

[end]

3. Within this method, retrieve the Tables instance from the Tables property of the Database property of the ReportDocument parameter.

   Note Tables is an indexed class that contains instances of the Table class.

[Visual Basic]

```
Dim myTables As Tables = myReportDocument.Database.Tables
```

[end]

[C#]

```
Tables tables = reportDocument.Database.Tables;
```

[end]

4. Create a foreach loop that loops through each Table instance in the Tables indexed class instance.

   Note You must include the full namespace path to Table class, to distinguish it from the Table class of the System.Web.UI.WebControls namespace.

[Visual Basic]

```
For Each myTable As CrystalDecisions.CrystalReports.Engine.Table In
myTables


Next
```

[end]

[C#]

```
foreach(CrystalDecisions.CrystalReports.Engine.Table table in tables)
{
}
```

[end]

5. Within the foreach loop, retrieve the TableLogonInfo instance from the LogOnInfo property of the Table instance.

[Visual Basic]

```
Dim myTableLogonInfo As TableLogOnInfo = myTable.LogOnInfo
```

[end]

[C#]

```
TableLogOnInfo tableLogonInfo = table.LogOnInfo;
```

[end]

6. Within the foreach loop, set the ConnectionInfo property of TableLogonInfo to the ConnectionInfo parameter.

[Visual Basic]

```
myTableLogonInfo.ConnectionInfo = myConnectionInfo
```

[end]

[C#]

```
tableLogonInfo.ConnectionInfo = connectionInfo;
```

[end]

7. Within the foreach loop, pass the TableLogonInfo instance as a parameter to the ApplyLogonInfo method of the Table instance.

[Visual Basic]

```
myTable.ApplyLogOnInfo(myTableLogonInfo)
```

[end]

[C#]

```
table.ApplyLogOnInfo(tableLogonInfo);
```

[end]

This step procedure has created a method to set the logon for the database. However, you must modify the `ConfigureCrystalReports()` method to address this method, for the report to be aware that it has database logon information.

Modifying the ConfigureCrystalReports() method requires two actions:

Configure the ConnectionInfo instance.

Call the SetDBLogonForReport() method.

To modify the ConfigureCrystalReports() method to address the database logon code

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
2. Within the line breaks, declare and instantiate the ConnectionInfo class.

   Note To make the ConnectionInfo class accessible, include an `"Imports" [Visual Basic]` or `"using" [C#]` statement at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

[Visual Basic]

```
Dim myConnectionInfo As ConnectionInfo = New ConnectionInfo()
```

[end]

[C#]

```
ConnectionInfo connectionInfo = new ConnectionInfo();
```

[end]

3. Set the DatabaseName, UserID, and Password properties of the ConnectionInfo instance.

Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

In the code that you write, replace the sample *1234* password (shown below) with your own password.

[Visual Basic]

```
myConnectionInfo.DatabaseName = "Northwind"
myConnectionInfo.UserID = "limitedPermissionAccount"
myConnectionInfo.Password = "1234"
```

[end]

[C#]

```
connectionInfo.DatabaseName = "Northwind";
connectionInfo.UserID = "limitedPermissionAccount";
connectionInfo.Password = "1234";
```

[end]

4. Enter a call to the `SetDBLogonForReport()` method, by passing in the ConnectionInfo instance and the NorthwindCustomers report.

[Visual Basic]

```
SetDBLogonForReport(myConnectionInfo, northwindCustomersReport)
```

[end]

[C#]

```
SetDBLogonForReport(connectionInfo, northwindCustomersReport);
```

[end]

This is followed by the original code that binds the report to the CrystalReportViewer control.

You are now ready to build and run your project. The report should load properly, because you have added code to log on to the database.

To test the loading of the NorthwindCustomers report

1. From the **Build** menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you learn how to change the database location at runtime.

Adding Ability to Change Database Location at Runtime

In this section, you learn how to change the database location at runtime. This requires only a minor modification to the ConnectionInfo instance.

To change the database location at runtime

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the line that declares and instantiates the ConnectionInfo class.
2. Within the line breaks, set the ServerName property of the ConnectionInfo instance.

   Note In the code that you write, replace the sample server name *DevDatabase* (shown below) with the name of your server.

[Visual Basic]

```
myConnectionInfo.ServerName = "DevDatabase"
```

[end]

[C#]

```
connectionInfo.ServerName = "DevDatabase";
```

[end]

You are now ready to build and run your project. The report should redirect to the alternate database server at runtime.

To test that the report can be reset to an alternate database server at runtime

1. From the Build menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully set your code to change the database location at runtime. In this example, it is the same database server, the only difference is that you have called it explicitly by name.

However, you can now change this database server name string to any other database server that contains the Northwind database.

Note You may wish to work further with this tutorial by adding a subreport into your report and configuring that subreport to logon to the secure SQL Server database. If so, proceed to the next tutorial which modifies your current tutorial: Logging onto a Secure SQL Server Database with a Subreport.

To learn about logging on to a SQL Server database with enhanced API features, continue to Addendum: Enhancements to the Database Logon Code.

Addendum: Enhancements to the Database Logon Code

If you have installed Visual Studio 2005 or Crystal Reports Developer you have access to the enhanced API for logging on to a secure SQL Server database. The Crystal Reports Developer API helps minimize the amount of code that is needed to log on to the database.

In the previous procedures, you learned to create the SetDBLogonForReport() helper method, which uses a foreach loop to set the ConnectionInfo property of each table in the Crystal report.

In this tutorial, you learn how to delete the helper method and add code to use either the DataSourceConnections class from CrystalDecisions.Shared namespace or the SetDatabaseLogon() method from the ReportDocument class.

The DataSourceConnections class is an ArrayList that contains the ConnectionInfo instances for every connection that is used by the Crystal report. You can retrieve the ConnectionInfo instances at a specified index, and then call SetLogon() or SetConnection() to pass the logon information to the report.

The SetLogon() method allows you to set the user name and password. This method uses the default server and database that you have specified in the report. The SetConnection() method allows you to set the server name, database name, user name, and password.

To use the new API code, you must complete the instructions in Creating a Report Connected to a Secure SQL Server Database and Binding the Report Without Logon Code.

Then, you can use one of the following enhanced API methods:

Using the DataSourceConnections Class for Database Logon

Using the SetDatabaseLogon() Method of the ReportDocument Class

If you have completed all the procedures in Logging onto a Secure SQL Server Database Using SQL Authentication, you must first delete certain lines of code that are shown in Modifying the Project for Database Logon, before you can use one of the enhanced API methods.

Modifying the Project for Database Logon

If you have completed all the procedures in Logging onto a Secure SQL Server Database Using SQL Authentication, you must first delete certain lines of code that are shown in the following procedure.

To modify the project for using the enhanced Crystal Reports API

1. Open the completed project for this tutorial.
2. Open the Web or Windows Form.
3. From the **View** menu, click **Code**.
4. Delete the `SetDBLogonForReport()` helper method.
5. Within the `ConfigureCrystalReports()` method, delete the following lines of code:
   a) Delete the code that declares an instance of the ConnectionInfo class.
   b) Delete the code that uses the ServerName, DatabaseName, UserID and Password properties from the ConnectionInfo class.
   c) Delete the call to the `SetDBLogonForReport()` method.

Now, the `ConfigureCrystalReports()` method has two lines of code.

[Visual Basic]

```
Private Sub ConfigureCrystalReports()
    northwindCustomersReport = new NorthwindCustomers()
    myCrystalReportViewer.ReportSource = northwindCustomersReport
End Sub
```

[end]

[C#]

```
private void ConfigureCrystalReports()
{
    northwindCustomersReport = new NorthwindCustomers();
    crystalReportViewer.ReportSource = northwindCustomersReport;
}
```

[end]

You can now choose to use one of the following enhanced API methods:

Using the DataSourceConnections Class for Database Logon

Using the SetDatabaseLogon() Method of the ReportDocument Class

Using the DataSourceConnections Class for Database Logon

In this section, you learn how to add two lines of code to log on to a SQL Server database. You must retrieve the DataSourceConnections instance, and then set the database logon information through use of SetLogon() or SetConnection().

Prerequisites:

You must create a project that is based on the instructions in Creating a Report Connected to a Secure SQL Server Database and Binding the Report Without Logon Code.

Or, you must create a project that is based on the instructions in Modifying the Project for Database Logon.

If you want to use the default server and database, call the SetLogon() method with your user name and password.

Otherwise, if you want to change the server or database, call the SetConnection() method with your server name, database name, user name and password.

To use the SetLogon() method of the DataSourceConnections class

1. Between the two lines of code within the `ConfigureCrystalReports()`, retrieve the DataSourceConnections instance from the DataSourceConnections property of the NorthwindCustomers instance.

[Visual Basic]

```
Dim myDataSourceConnections As DataSourceConnections =
northwindCustomersReport.DataSourceConnections
```

[end]

[C#]

```
DataSourceConnections dataSourceConnections =
northwindCustomersReport.DataSourceConnections;
```

[end]

2. Retrieve the IConnectionInfo at index 0 of the DataSourceConnections instance.

[Visual Basic]

```
Dim myConnectInfo As IConnectionInfo = myDataSourceConnections(0)
```

[end]

[C#]

```
IConnectionInfo connectInfo = dataSourceConnections[0];
```

[end]

3. Call the `SetLogon()` method with your user name and password

   Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

   In the code that you write, replace the sample *1234* password (shown below) with your own password.

[Visual Basic]

```
myConnectInfo.SetLogon("limitedPermissionAccount", "1234")
```

[end]

[C#]

```
connectInfo.SetLogon("limitedPermissionAccount", "1234");
```

[end]

To use the SetConnection() method of the DataSourceConnections class

1. Between the two lines of code within the `ConfigureCrystalReports()`, retrieve the DataSourceConnections instance from the DataSourceConnections property of the NorthwindCustomers instance.

[Visual Basic]

```
Dim myDataSourceConnections As DataSourceConnections =
northwindCustomersReport.DataSourceConnections
```

[end]

[C#]

```
DataSourceConnections dataSourceConnections =
northwindCustomersReport.DataSourceConnections;
```

[end]

2. Retrieve the IConnectionInfo at index 0 of the DataSourceConnections instance.

[Visual Basic]

```
    Dim myConnectInfo As IConnectionInfo = myDataSourceConnections(0)
```

[end]

[C#]

```
    IConnectionInfo connectInfo = dataSourceConnections[0];
```

[end]

3. Call the `SetConnection()` method with your server name, database name, user name and password.

   Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

   In the code that you write, replace the sample *1234* password (shown below) with your own password.

[Visual Basic]

```
myConnectInfo.SetConnection("ServerName", "Northwind",
"limitedPermissionAccount", "1234")
```

[end]

[C#]

```
connectInfo.SetConnection("ServerName", "Northwind",
"limitedPermissionAccount", "1234");
```

[end]

You are now ready to build and run the project, to log on to the secure SQL Server database.

To learn about another enhanced API method, see Using the SetDatabaseLogon() Method of the ReportDocument Class.

Using the SetDatabaseLogon() Method of the ReportDocument Class

In this section, you learn how to call the SetDatabaseLogon() method in the ConfigureCrystalReports() method to log on to the database. The SetDatabaseLogon() method is overloaded into two sets of parameters:

User name and password.

User name, password, server name, and password.

The use of this method minimizes the code that you need. However, this method does not allow you to change servers or databases.

Prerequisites:

You must create a project that is based on the instructions in Creating a Report Connected to a Secure SQL Server Database and Binding the Report Without Logon Code.

Or, you must create a project that is based on the instructions in Modifying the Project for Database Logon.

To use the SetDatabaseLogon() method of the ReportDocument class

Between the two lines of code within the `ConfigureCrystalReports()`, call the `SetDatabaseLogon()` method of the NorthwindCustomers instance, and pass in one of the following sets of parameters:

1. Call the `SetDatabaseLogon()` method with your user name and password.

   Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

   In the code that you write, replace the sample *1234* password (shown below) with your own password.

[Visual Basic]

```
northwindCustomersReport.SetDatabaseLogon("limitedPermissionAccount"
, "1234")
```

[end]

[C#]

```
northwindCustomersReport.SetDatabaseLogon("limitedPermissionAccount"
, "1234");
```

[end]

2. Or call the `SetDatabaseLogon()` method with your server name, database name, user name and password.

   Note This method does not change the server or database. You are restricted to only the default server and database that is specified within the report.

   For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

   In the code that you write, replace the sample *1234* password (shown below) with your own password.

[Visual Basic]

```
northwindCustomersReport.SetDatabaseLogon("limitedPermissionAccount"
, "1234", "ServerName", "Northwind")
```

[end]

[C#]

```
northwindCustomersReport.SetDatabaseLogon("limitedPermissionAccount"
, "1234", "ServerName", "Northwind");
```

[end]

You are now ready to build and run the project, to log on to the secure SQL Server database.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_DBLogon

C# Windows project: CS_Win_RDObjMod_DBLogon

Visual Basic Web Site: VB_Web_RDObjMod_DBLogon

Visual Basic Windows project: VB_Win_RDObjMod_DBLogon

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Logging onto a Secure SQL Server Database Using Integrated Security

Logging onto a Secure SQL Server Database Using Integrated Security

Introduction

In this tutorial, you learn how to add logon code to display a report that contains information from a secure SQL Server database.

To log onto a secure SQL Server database, you use classes from the ReportDocument object model.

The ReportDocument object has a Database property that returns a Database instance. This Database instance contains the database information for the report, including a Tables property that returns a Tables indexed class instance. Individual Table instances can then be retrieved from the Tables indexed class.

Logon occurs at the granular level of each Table instance, which must be granted individual access to the secure SQL Server. This is done by placing logon information into a ConnectionInfo instance, and then, within a for loop, applying that ConnectionInfo instance to the ConnectionInfo Property of each Table instance.

The properties of the ConnectionInfo class include the following:

ServerName

DatabaseName

UserID (not used in this tutorial)

Password (not used in this tutorial)

IntegratedSecurity

Note If you wish to work through a tutorial that uses SQL Authentication (and therefore uses the UserID and Password properties rather than the Integrated Security property), see Logging onto a Secure SQL Server Database Using SQL Authentication.

If you choose only to set the DatabaseName, and IntegratedSecurity properties, you will be logged onto the default server and database specified within the report. However, if you choose to assign an alternate ServerName property, you can redirect the report to a different server at runtime.

You begin by creating a report containing data from a secure SQL server database.

You can complete this tutorial by using classes of the CrystalReportViewer object model; however, the ReportDocument object model is recommended.

Creating a Report Connected to a Secure SQL Server Database Using Integrated Security

To begin, create a report that draws its information from the Northwind database.

Note Northwind is a sample database provided with SQL Server.

Some setup is required as a prerequisite to this tutorial.

Prerequisite Database Setup

1. SQL Server configuration:

   If you have SQL Server (or the OEM version, MSDE) installed, it must be configured to accept Windows Authentication for this tutorial.

   If you do not have SQL Server (or the OEM version, MSDE) installed, you must install MSDE with the (default) Windows Authentication to work with this tutorial.
2. The Northwind database provided with SQL Server must be installed and verified that it accepts Windows Authentication.

To install MSDE with Windows Authentication, or the Northwind database, go to the following sections from Appendix: System Setup in this documentation:

Appendix: MSDE Installation with Windows or SQL Server Authentication

Appendix: Northwind Database Installation

Once you have configured SQL Server and the Northwind database according to the sections above, you are ready to create a report that draws its information securely from a Northwind database.

To create a report with secure data from the Northwind database

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the **Crystal Report** template.
3. In the **Name** field, enter the name "NorthwindCustomers.rpt" and click **Open**.

   Note If you have not registered before, you may be asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
4. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
5. In the **Choose an Expert** panel, select **Standard**. Click **OK**.

   The Standard Report Creation Wizard window appears.
6. In the **Available Data Sources** panel, expand the **Create New Connection** folder.
7. From the subfolder that opens, expand the **OLE DB (ADO)** folder.

   The OLE DB (ADO) window appears.
8. Select **Microsoft OLE DB Provider for SQL Server** and click **Next**.
9. Enter the name of your database server into the **Server** field.
10. Check the **Integrated Security** checkbox.
11. From the **Database** drop down list, select "Northwind."
12. Click **Finish**.

    The OLE DB folder is now expanded, showing your database server and within it, the Northwind database.

13. Expand the nodes **Northwind**, **dbo**, and **Tables**, and then select the **Customers** table.
14. Click the **>** symbol to move the table into the **Selected Tables** panel, then click **Next**.
15. Hold down the Ctrl key while clicking **CompanyName**, **ContactName** and **City**.
16. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click **Next**.
17. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, then click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

    The NorthwindCustomers report is created and loaded into the main window of Visual Studio.

You are now ready to bind the report to the CrystalReportViewer control and set the database logon programmatically.

Binding the Report

In Appendix: Project Setup, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous step, you have added a NorthwindCustomers report to the project.

In this section, you instantiate the NorthwindCustomers report and bind it to the CrystalReportViewer control. Then you test whether the report displays correctly when current values have not been set for its parameter field.

You can instantiate and bind the report in two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the report as a non-embedded report.

To instantiate the NorthwindCustomers report as an embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the NorthwindCustomers report wrapper class, using the variable name northwindCustomersReport. Set its access modifier to private.

[Visual Basic]

```
Private northwindCustomersReport As NorthwindCustomers
```

[end]

[C#]

```
private NorthwindCustomers northwindCustomersReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
northwindCustomersReport = New NorthwindCustomers()
```

[end]

[C#]

```
northwindCustomersReport = new NorthwindCustomers();
```

[end]

5. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer control to the instantiated report class (variable name: northwindCustomersReport).

[Visual Basic]

```
myCrystalReportViewer.ReportSource = northwindCustomersReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = northwindCustomersReport;
```

[end]

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to log onto the database.

To instantiate the NorthwindCustomers report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, using the variable name northwindCustomersReport. Set its access modifier to private.

[Visual Basic]

```
Private northwindCustomersReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument northwindCustomersReport;
```

[end]

   Note The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
northwindCustomersReport = New ReportDocument()
```

[end]

[C#]

```
northwindCustomersReport = new ReportDocument();
```

[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("NorthwindCustomers.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("NorthwindCustomers.rpt");
```

[end]

   For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

   Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"NorthwindCustomers.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"NorthwindCustomers.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
northwindCustomersReport.Load(reportPath)
```

[end]

[C#]

```
northwindCustomersReport.Load(reportPath);
```

[end]

7. Bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = northwindCustomersReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = northwindCustomersReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: northwindCustomersReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to log onto the database.

To test the loading of the NorthwindCustomers report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The NorthwindCustomers report does not display, because the database logon code has not been added.

   Note Results may vary, depending on the version of Crystal Reports that you use. For example, if you have Crystal Reports Developer installed, a form appears and requests that you provide database logon information for that report. This is a new feature of Crystal Reports Developer. If you are running a previous version of Crystal Reports, an exception is thrown. In either case, you must follow the next step procedure to create a fully functional application.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Adding the Report Logon Code

You are now ready to add the logon code to the code-behind class. You begin by creating a private helper method, `SetDBLogonForReport()`.

To create and code the SetDBLogonForReport() method

1. Return to the code-behind class for this Web or Windows Form.
2. At the bottom of the class, create a new private method named `SetDBLogonForReport()` with two parameters, ConnectionInfo and ReportDocument.

[Visual Basic]

```
Private Sub SetDBLogonForReport(ByVal myConnectionInfo As ConnectionInfo,
ByVal myReportDocument As ReportDocument)


End Sub
```

[end]

[C#]

```
private void SetDBLogonForReport(ConnectionInfo connectionInfo,
ReportDocument reportDocument)
{
}
```

[end]

3. Within this method, retrieve the Tables instance from the Tables property of the Database property of the ReportDocument parameter.

   Note Tables is an indexed class that contains instances of the Table class.

[Visual Basic]

```
Dim myTables As Tables = myReportDocument.Database.Tables
```

[end]

[C#]

```
Tables tables = reportDocument.Database.Tables;
```

[end]

4. Create a foreach loop that loops through each Table instance in the Tables indexed class instance.

   Note You must include the full namespace path to Table class, to distinguish it from the Table class of the System.Web.UI.WebControls namespace.

[Visual Basic]

```
For Each myTable As CrystalDecisions.CrystalReports.Engine.Table In
myTables


Next
```

[end]

[C#]

```
foreach(CrystalDecisions.CrystalReports.Engine.Table table in tables)
{
}
```

[end]

5. Within the foreach loop, retrieve the TableLogonInfo instance from the LogOnInfo property of the Table instance.

[Visual Basic]

```
Dim myTableLogonInfo As TableLogOnInfo = myTable.LogOnInfo
```

[end]

[C#]

```
TableLogOnInfo tableLogonInfo = table.LogOnInfo;
```

[end]

6. Within the foreach loop, set the ConnectionInfo property of TableLogonInfo to the ConnectionInfo parameter.

[Visual Basic]

```
myTableLogonInfo.ConnectionInfo = myConnectionInfo
```

[end]

[C#]

```
tableLogonInfo.ConnectionInfo = connectionInfo;
```

[end]

7. Within the foreach loop, pass the TableLogonInfo instance as a parameter to the ApplyLogonInfo method of the Table instance.

[Visual Basic]

```
myTable.ApplyLogOnInfo(myTableLogonInfo)
```

[end]

[C#]

```
table.ApplyLogOnInfo(tableLogonInfo);
```

[end]

This step procedure has created a method to set the logon for the database. However, you must modify the `ConfigureCrystalReports()` method to address this method, for the report to be aware that it has database logon information.

Modifying the ConfigureCrystalReports() method requires two actions:

Configure the ConnectionInfo instance.

Call the SetDBLogonForReport() method.

To modify the ConfigureCrystalReports() method to address the database logon code

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
2. Within the line breaks, declare and instantiate the ConnectionInfo class.

   Note To make the ConnectionInfo class accessible, include an `"Imports" [Visual Basic]` or `"using" [C#]` statement at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

[Visual Basic]

```
Dim myConnectionInfo As ConnectionInfo = New ConnectionInfo()
```

[end]

[C#]

```
ConnectionInfo connectionInfo = new ConnectionInfo();
```

[end]

3. Set the DatabaseName and IntegratedSecurity properties of the ConnectionInfo instance.

[Visual Basic]

```
myConnectionInfo.DatabaseName = "Northwind"
myConnectionInfo.IntegratedSecurity = True
```

[end]

[C#]

```
connectionInfo.DatabaseName = "Northwind";
connectionInfo.IntegratedSecurity = true;
```

[end]

4. Enter a call to the `SetDBLogonForReport()` method, by passing in the ConnectionInfo instance and the NorthwindCustomers report.

[Visual Basic]

```
SetDBLogonForReport(myConnectionInfo, northwindCustomersReport)
```

[end]

[C#]

```
SetDBLogonForReport(connectionInfo, northwindCustomersReport);
```

[end]

This is followed by the original code that binds the report to the CrystalReportViewer control.

You are now ready to build and run your project. The report should load properly, because you have added code to log on to the database.

To test the loading of the NorthwindCustomers report

1. From the **Build** menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you learn how to change the database location at runtime.

Adding Ability to Change Database Location at Runtime

In this section, you learn how to change the database location at runtime. This requires only a minor modification to the ConnectionInfo instance.

To change the database location at runtime

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the line that declares and instantiates the ConnectionInfo class.
2. Within the line breaks, set the ServerName property of the ConnectionInfo instance.

   Note In the code that you write, replace the sample server name *DevDatabase* (shown below) with the name of your server.

[Visual Basic]

```
myConnectionInfo.ServerName = "DevDatabase"
```

[end]

[C#]

```
connectionInfo.ServerName = "DevDatabase";
```

[end]

You are now ready to build and run your project. The report should redirect to the alternate database server at runtime.

To test that the report can be reset to an alternate database server at runtime

1. From the Build menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Configuration with the SetConnection() method

In this section, you learn how to apply all three changes (to the server name, database name, and integrated security setting) using the `SetConnection()` method. This requires only a minor modification to the ConnectionInfo instance.

To change to the SetConnection() method

1. In the `ConfigureCrystalReports()` method, after the line that declares and instantiates the ConnectionInfo class, delete the lines of code that assign the ServerName, DatabaseName, and IntegratedSecurity properties.
2. Enter a new line of code in which you call the `SetConnection()` method of the ConnectionInfo instance, passing to this method the server name, database name and integrated security setting.

[Visual Basic]

```
myConnectionInfo.SetConnection("DevDatabase", "Northwind", True)
```

[end]

[C#]

```
connectionInfo.SetConnection("DevDatabase", "Northwind", true);
```

[end]

You are now ready to build and run your project.

To test the SetConnection() method

1. From the Build menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully set your code to change the database location at runtime. In this example, it is the same database server, the only difference is that you have called it explicitly by name.

However, you can now change this database server name string to any other database server that contains the Northwind database.

Note You may wish to work further with this tutorial by adding a subreport into your report and configuring that subreport to logon to the secure SQL Server database. If

so, proceed to the tutorial which modifies your current tutorial: Logging onto a Secure SQL Server Database with a Subreport.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_DBLogonIntegratedSecurity

C# Windows project: CS_Win_RDObjMod_DBLogonIntegratedSecurity

Visual Basic Web Site: VB_Web_RDObjMod_DBLogonIntegratedSecurity

Visual Basic Windows project: VB_Win_RDObjMod_DBLogonIntegratedSecurity

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Logging onto a Secure SQL Server Database with a Subreport

Logging onto a Secure SQL Server Database with a Subreport

Introduction

In this tutorial, you explore an additional complication to Logging onto a Secure SQL Server Database: what if the report that requires secure SQL Server database logon contains a subreport?

To do this tutorial, you must complete the previous tutorial, Logging onto a Secure SQL Server Database.

In the previous tutorial, Logging onto a Secure SQL Server Database, you learned how to display a report that requires secure SQL Server Database logon, by writing code that passes logon information to the report at runtime.

In this tutorial, you learn how to meet logon requirements for the subreport.

You need to make two modifications to the project you have created in Logging onto a Secure SQL Server Database:

You add a subreport into the original report.

This subreport addresses the Orders table of the Northwind database. The Orders table is related to the Customers table that is used by a CustomerID foreign key in the previous tutorial.

You add a new method.

The method retrieves subreports out of the main report, and then passes each subreport to the `SetDBLogonForReport()` helper method.

You can also complete this tutorial with classes of the CrystalReportViewer object model, although the ReportDocument object model is preferred.

To build this tutorial with the CrystalReportViewer object model, see Logging onto a Secure SQL Server Database Using the CrystalReportViewer Object Model.

Adding a Subreport to the Original Report

You begin by adding a subreport to the original report.

To add a subreport

1. Open the project you created in the previous tutorial, Logging onto a Secure SQL Server Database.
2. From **Solution Explorer**, double-click the **NorthwindCustomers** report to open it.
3. Right-click the **Details** gray bar and select **Insert Section Below**.
4. Right-click within the new **Details b** section that you have created, point to **Insert**, and then click **Subreport**.

   A gray square appears around the mouse cursor.
5. Drag the gray rectangle into the new **Details b** section, and then click to release.
6. In the **Insert Subreport** dialog box, on the **Subreport** tab, select **Create a subreport with the Report Wizard**.

Note The Insert Subreport dialog box includes other options that allow you to choose an existing report and on-demand subreports.

7. In the **New report name** field, type "CustomerOrders."
8. Click **Report Wizard...**
9. In the **Standard Report Creation Wizard** dialog box, in the **Available Data Sources** panel, expand the **Create New Connection** folder.
10. Expand the **OLE DB (ADO)** folder.

    The folder contains the database server, which was configured for the report when the report was created.

    Note If the server is not displayed, follow the instructions in the previous tutorial to log onto the SQL Server database.
11. Expand the nodes **Northwind**, **dbo**, and **Tables**.
12. Select the **Orders** table and click the **>** symbol to move the Orders table into the **Select Tables** panel, and then click **Next**.
13. From the **Available Fields** panel, select **Order ID**, **Order Date**, **Shipped Date**, and **Ship Name**.
14. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Finish**.
15. In the **Insert Subreport** dialog box, select the **Link** tab.
16. In the panel **Container Report field(s) to link to**, in the list **Available fields**, expand the **Customers** table, select **CustomerID**, and then click the **>** symbol.
17. In the **Customers.CustomerID field link** panel that appears, leave the default selections unchanged.

    These parameter and data selections auto-generate a relationship between the main report and the subreport.
18. Click **OK**.

    The new subreport, CustomerOrders, is displayed within the Details b section of the Main report.

    Note Adding a subreport to the Details section means the subreport displays for every row, which adds a performance cost to your report. If you do not need subreport information with that level of granularity, place the subreport in a Group section rather than a Details section.

You are now ready to verify the settings in the subreport.

To verify the settings in the subreport

1. In the report Details section, double-click the **CustomerOrders** subreport to view it.

   At the bottom of the designer view, navigation buttons appear for both the Main Report and the CustomerOrders subreport.
2. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click **Toggle Field View**.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.
3. In the **Field Explorer**, expand **Parameter Fields**.

4. Verify that the parameter field **Pm-Customers.CustomerID** was auto-generated when the subreport was linked.
5. On the toolbar, click **Select Expert**.
6. In the **Select Expert** dialog box, verify that the criteria **Orders.CustomerID is equal to {?Pm-Customers.CustomerID}** is set, and then click **OK**.
7. From the **File** menu, select **Save All**.

   You have successfully added a CustomerOrders subreport to the NorthwindCustomers report. In the next section you add code to set secure SQL Server Database logon information for all subreports that are found within the main report.

Adding the Subreport Logon Code

You are now ready to add the logon code for the subreport to the code-behind class. To begin, you create a private helper method named `SetDBLogonForSubreports()`.

To create and code the SetDBLogonForSubreports() method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a new private method named `SetDBLogonForSubreports()` with two parameters, ConnectionInfo and ReportDocument.

[Visual Basic]

```
Private Sub SetDBLogonForSubreports(ByVal myConnectionInfo As
ConnectionInfo, ByVal myReportDocument As ReportDocument)


End Sub
```

[end]

[C#]

```
private void SetDBLogonForSubreports(ConnectionInfo connectionInfo,
ReportDocument reportDocument)
{
}
```

[end]

4. Within this method, retrieve the Sections instance from the Sections property of the ReportDefinition property of the ReportDocument parameter.

   Note Sections is an indexed class that contains instances of the Section class.

[Visual Basic]

```
Dim mySections As Sections = myReportDocument.ReportDefinition.Sections
```

[end]

[C#]

```
Sections sections = reportDocument.ReportDefinition.Sections;
```

[end]

5. Create a foreach loop that loops through each Section instance in the Sections indexed class instance.

[Visual Basic]

```
For Each mySection As Section In mySections


Next
```

[end]

[C#]

```
foreach(Section section in sections)
{
}
```

[end]

6. Within the foreach loop, retrieve the ReportObjects instance from the ReportObjects property of the Section instance.

[Visual Basic]

```
Dim myReportObjects As ReportObjects = mySection.ReportObjects
```

[end]

[C#]

```
ReportObjects reportObjects = section.ReportObjects;
```

[end]

7. Within the foreach loop, create a new, nested foreach loop that loops through each ReportObject instance in the ReportObjects indexed class instance.

[Visual Basic]

```
For Each myReportObject As ReportObject In myReportObjects


Next
```

[end]

[C#]

```
foreach(ReportObject reportObject in reportObjects)
{
}
```

[end]

8. Within the nested foreach loop, create a condition block that tests whether the Kind property of the ReportObject instance is equal to the SubreportObject selection of the ReportObjectKind enumeration.

[Visual Basic]

```
If myReportObject.Kind = ReportObjectKind.SubreportObject Then


End If
```

[end]

[C#]

```
if(reportObject.Kind == ReportObjectKind.SubreportObject)
{
}
```

[end]

9. Within the conditional block, cast the ReportObject instance to a SubreportObject instance.

[Visual Basic]

```
Dim mySubreportObject As SubreportObject = CType(myReportObject,
SubreportObject)
```

[end]

[C#]

```
SubreportObject subreportObject = (SubreportObject)reportObject;
```

[end]

10. Still within the conditional block, declare a new instance of ReportDocument as a subReportDocument variable and populate it by calling the `OpenSubreport()` method of the SubreportObject instance.

[Visual Basic]

```
Dim subReportDocument As ReportDocument =
mySubreportObject.OpenSubreport(mySubreportObject.SubreportName)
```

[end]

[C#]

```
ReportDocument subReportDocument =
subreportObject.OpenSubreport(subreportObject.SubreportName);
```

[end]

11. Finally, within the conditional block, call the original `SetDBLogonForReport()` method and pass to it the ConnectionInfo parameter and the new subReportDocument variable.

[Visual Basic]

```
SetDBLogonForReport(myConnectionInfo, subReportDocument)
```

[end]

[C#]

```
SetDBLogonForReport(connectionInfo, subReportDocument);
```

[end]

This step procedure has created a method to retrieve all possible subreports as ReportDocument instances. Each ReportDocument instance of subreport is then passed to the original method for setting the database logon to a Secure SQL Server database.

However, you must now modify the `ConfigureCrystalReports()` method to address this method, in order for subreports to have their database logon information processed.

To modify the ConfigureCrystalReports() method to address the DB Logon code for subreports

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
2. Within the line breaks, enter a call to the `SetDBLogonForSubreports()` method, by passing in the ConnectionInfo instance and the NorthwindCustomers report.

[Visual Basic]

```
SetDBLogonForSubreports(myConnectionInfo, northwindCustomersReport)
```

[end]

[C#]

```
SetDBLogonForSubreports(connectionInfo, northwindCustomersReport);
```

[end]

This is followed by the original code, which binds the report to the CrystalReportViewer control.

You are now ready to build and run your project. It is expected that the report loads successfully, including its subreport, because there is now code written to log on any subreports to the database.

To test the loading of the NorthwindCustomers report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The NorthwindCustomers report, including its new CustomerOrders subreport displays successfully.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully modified your tutorial project to use a report that contains a subreport and change the database location at runtime for both the report and its subreport.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_DBLogonSubrpt

C# Windows project: CS_Win_RDObjMod_DBLogonSubrpt

Visual Basic Web Site: VB_Web_RDObjMod_DBLogonSubrpt

Visual Basic Windows project: VB_Win_RDObjMod_DBLogonSubrpt

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Reading and Setting Discrete Parameters

Reading and Setting Discrete Parameters

Introduction

In this tutorial, you learn how to create a report whose data can be filtered based on a discrete parameter.

A discrete parameter is a single value, as opposed to a range parameter, which refers to a range of values. Text items (such as cities) are usually accessed through discrete parameters. Numeric items (such as employee salaries) are usually accessed through range parameters.

In this tutorial, you set a value for a discrete parameter to view a customer report that is based on one field. The report displays only those customers who live in the cities you have selected from a city list. The city list is derived from the default values for the City parameter. Those default values for the City parameter are encapsulated within the report.

To begin, you create a customer report with a city parameter. The report derives its data from the sample database that is shipped with Crystal Reports for Visual Studio. When you instantiate the report in the code, you create an ArrayList that contains city names (Paris, Tokyo) and pass that ArrayList instance to a helper method that sets those city names as the current values for the city parameter. You then bind the report to the CrystalReportViewer control and view the report with only those customers that live in Paris and Tokyo displayed.

In the next part of the tutorial, you do the following:

Create a method that gets all the default values and returns them in an ArrayList.

Add a ListBox control to the form and populate it from the ArrayList.

Add a Button control to redisplay the report based on ListBox selections.

In the final part of the tutorial, you code the button click event to retrieve any selected items from the ListBox control and set those to be the current values for the city parameter. The report redisplays and shows only those customers who live in the cities that have been selected within the ListBox control.

This tutorial can also be completed with classes of the CrystalReportViewer object model, although the ReportDocument object model is preferred.

To build this tutorial with the CrystalReportViewer object model, see Reading and Setting Discrete Parameters using the CrystalReportViewer object model.

Creating a Report with Parameters

To begin, create a report that draws its information from the sample Microsoft Access database that ships with Crystal Reports.

Note Xtreme.mdb is the sample database that is provided with Crystal Reports. To locate xtreme.mdb on your hard drive for your version of Crystal Reports, see Appendix: Location of Xtreme Sample Database. You need to connect to the database through its ODBC DSN entry. To learn the name of this entry for your version of Crystal Reports, see Appendix:ODBC DSN Entry for Xtreme Sample Database.

To create a report with parameters

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the template named "Crystal Report."
3. In the **Name** field, enter the name "CustomersByCity.rpt" and click **Add**.

   Note In Visual Studio .NET 2002 or 2003, the button is named Open.
4. If you have not registered before, you are asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
5. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
6. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
7. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to the full version, the Create New Connection folder does not exist; the contents are shown at the top level.
8. From the subfolder that opens, expand the **ODBC (RDO)** folder.
9. In the ODBC (RDO) window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
10. Expand the **Tables** node and select the **Customer** table.
11. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
12. Expand the **Customer** table, then CTRL-click **Customer Name**, **Contact Title**, **Address1**, **Contact Last Name** and **City**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click the **Next** button.
14. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click the **Finish** button.

    The CustomersByCity report is created and loaded into the main window of Visual Studio.

You are now ready to add a parameter named City and populate it with default values.

To add a City parameter

The Field Explorer must be visible, because it provides access to the various features of the report, including parameters.

1. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click **Toggle Field View**.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.
2. In the **Field Explorer**, right-click **Parameter Fields** and select **New...**
3. In the **Create Parameter Field** dialog box:

   Set the **Name** to "City."

   Set the **Prompting Text** to "Select one or more cities."

   Set **Value Type** to **String**

   Select **Allow multiple values**.

   Select **Discrete value(s)**.

   Click **Default Values...**

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to the full version, this button is named Set Default Values.
4. In the **Set Default Values** dialog box:

   Set the **Browse table** to "Customer."

   Set the **Browse field** to "City."

   Click **>>** (the double right arrow) to move the entire cities list into the **Default Values** list.
5. Click **OK** to close the **Set Default Values** dialog box.
6. Click **OK** to close the **Create Parameter Field** dialog box.

You have just set the Default Values to contain a large list of cities. Later in this tutorial, you access this same list of default values programmatically, through the DefaultValues property of the ParameterFieldDefinition class.

You now use the Select Expert to set a formula that connects the City database column to your newly created City parameter field.

To connect the City parameter to the City database column

1. On the Crystal Reports toolbar, click **Select Expert**.
2. In the **Choose Field** dialog box, under **Report Fields**, select **Customer.City**, and then click **OK**.
3. In the **Select Expert** dialog box, within the **Customer.City** tab, set the dropdown list to "is equal to."
4. In the new dropdown list that appears to the right, select the first choice in the list, **{?City}**, and then click **OK**.

   Note This selection, {?City}, is the City parameter that you created earlier.
5. From the **File** menu, select **Save All**.

You are now ready to bind the report to the CrystalReportViewer control and set the city parameter with two initial values, Paris and Tokyo.

Binding the Report

When you followed the instructions in the section Appendix: Project Setup to prepare for this tutorial, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous steps, you added a CustomersByCity report to the project.

In this section, you instantiate the CustomersByCity report and bind it to the CrystalReportViewer control. Then you test whether the report displays correctly when current values have not been set for its parameter field.

You can instantiate and bind the report two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the report as a non-embedded report.

To instantiate the CustomersByCity report as an embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code** to view the code-behind class for this Web or Windows Form.
3. Add a new class-level declaration for the CustomersByCity report wrapper class, with the variable name customersByCityReport. Set its access modifier to private.

[Visual Basic]

```
Private customersByCityReport As CustomersByCity
```

[end]

[C#]

```
private CustomersByCity customersByCityReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
customersByCityReport = New CustomersByCity()
```

[end]

[C#]

```
customersByCityReport = new CustomersByCity();
```

[end]

5. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer control to the instantiated report class (variable name: customersByCityReport).

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customersByCityReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customersByCityReport;
```

[end]

Note The CrystalReportViewer control instance is accessible in the code because you added the control to your Web or Windows form. If Intellisense does not recognize the CrystalReportViewer control instance, verify that the CrystalReportViewer control has been added as a class-level declaration to this code-behind class.

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to set a value for the City parameter field.

To instantiate the CustomersByCity report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, with the variable name customersByCityReport. Set its access modifier to private.

[Visual Basic]

```
Private customersByCityReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument customersByCityReport;
```

[end]

Note The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (which you added during one of the procedures in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
customersByCityReport = New ReportDocument()
```

[end]

[C#]

```
customersByCityReport = new ReportDocument();
```

[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("CustomersByCity.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("CustomersByCity.rpt");
```

[end]

For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"CustomersByCity.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"CustomersByCity.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
customersByCityReport.Load(reportPath)
```

[end]

[C#]

```
customersByCityReport.Load(reportPath);
```

[end]

7. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customersByCityReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customersByCityReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: customersByCityReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to set a value for the City parameter field. You'll add a value for the City parameter field later in this tutorial.

To test the loading of the CustomersByCity report

1. From the **Build** menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The CustomersByCity report does not display. It displays after you add a value for the City parameter field later in this tutorial.

   Note Results may vary, depending on the version of Crystal Reports that you use. In more recent versions, you can see a form requesting that you provide parameter values for that report. In earlier versions, a "Missing parameter field current value" exception is thrown. In either case, you must add further code to create a fully functional application.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Setting Parameters Manually in Code

You are now ready to set two values ("Paris" and "Tokyo") into the City parameter field for the CustomersByCity report.

This involves some coding, which you can separate into the following processes:

You need a PARAMETER_FIELD_NAME constant to hold the "City" parameter field name, which is used several times throughout the code.

The code to add current values to the parameter is commonly used at two different locations in this tutorial; therefore, you create this code as a separate helper method.

Within the `ConfigureCrystalReports()` method, you add the "Paris" and "Tokyo" parameters to an ArrayList instance and pass in both the report and the ArrayList instance to the helper method to be processed.

To create a PARAMETER_FIELD_NAME constant

1. Return to the code-behind class for this Web or Windows Form.
2. At the class level, create a new string constant, PARAMETER_FIELD_NAME, and set its value to "City."

[Visual Basic]

```
Private Const PARAMETER_FIELD_NAME As String = "City"
```

[end]

[C#]

```
private const string PARAMETER_FIELD_NAME = "City";
```

[end]

You are now ready to create the helper method that adds current values to the parameter in the report.

To create a helper method that adds current values to the parameter in the report

1. Return to the code-behind class for this Web or Windows Form.
2. Above the class signature, add an "Imports" [Visual Basic] or "using" [C#] declaration to the top of the class for the System.Collections namespace (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Collections
```

[end]

[C#]

```
using System.Collections;
```

[end]

Note This declaration is needed, to access the ArrayList class.

3. At the bottom of the class, create a new private method named SetCurrentValuesForParameterField(), with two variables in the method signature: ReportDocument, and ArrayList.

Note Later in this tutorial, if you have used an embedded report, you pass your embedded report class into the ReportDocument method parameter. How is this possible? All embedded report classes in Crystal Reports inherit from the ReportDocument base class.

[Visual Basic]

```
Private Sub SetCurrentValuesForParameterField(ByVal myReportDocument As
ReportDocument, ByVal myArrayList As ArrayList)


End Sub
```

[end]

[C#]

```
private void SetCurrentValuesForParameterField(ReportDocument
reportDocument, ArrayList arrayList)
{

}
```

[end]

4. Within this method, declare and instantiate the ParameterValues that are indexed class as the variable currentParameterValues.

Note For the ParameterValues class to be accessible, you must have included an "Imports" [Visual Basic] or "using" [C#] declaration at the top of the

code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

[Visual Basic]

```
Dim currentParameterValues As ParameterValues = New ParameterValues()
```

[end]

[C#]

```
ParameterValues currentParameterValues = new ParameterValues();
```

[end]

5. Create a foreach loop to retrieve all of the submitted values (as type Object) from the ArrayList instance.

   Note In this method, you retrieve values from the ArrayList. Later you write code that adds values to the ArrayList.

[Visual Basic]

```
For Each submittedValue As Object In myArrayList


Next
```

[end]

[C#]

```
foreach(object submittedValue in arrayList)
{
}
```

[end]

6. Within the foreach loop, declare and instantiate the ParameterDiscreteValue class.

[Visual Basic]

```
Dim myParameterDiscreteValue As ParameterDiscreteValue = New
ParameterDiscreteValue()
```

[end]

[C#]

```
ParameterDiscreteValue parameterDiscreteValue = new
ParameterDiscreteValue();
```

[end]

7. Within the foreach loop, convert the submittedValue to string and pass it to the Value property of the ParameterDiscreteValue instance.

[Visual Basic]

```
myParameterDiscreteValue.Value = submittedValue.ToString()
```

[end]

[C#]

```
parameterDiscreteValue.Value = submittedValue.ToString();
```

[end]

8. Within the foreach loop, add the ParameterDiscreteValue instance into the currentParameterValues indexed class.

[Visual Basic]

```
currentParameterValues.Add(myParameterDiscreteValue)
```

[end]

[C#]

```
currentParameterValues.Add(parameterDiscreteValue);
```

[end]

This completes the code within the foreach loop. You place the remaining code (from the steps that follow) after the foreach loop.

9. Outside the foreach loop, retrieve the ParameterFieldDefinitions indexed class, which comes from the DataDefinition property of the ReportDocument instance.

[Visual Basic]

```
Dim myParameterFieldDefinitions As ParameterFieldDefinitions =
myReportDocument.DataDefinition.ParameterFields
```

[end]

[C#]

```
ParameterFieldDefinitions parameterFieldDefinitions =
reportDocument.DataDefinition.ParameterFields;
```

[end]

10. Retrieve the ParameterFieldDefinition instance from the ParameterFieldDefinitions indexed class that is based on the index entry of the PARAMETER_FIELD_NAME constant.

[Visual Basic]

```
Dim myParameterFieldDefinition As ParameterFieldDefinition =
myParameterFieldDefinitions(PARAMETER_FIELD_NAME)
```

[end]

[C#]

```
ParameterFieldDefinition parameterFieldDefinition =
parameterFieldDefinitions[PARAMETER_FIELD_NAME];
```

[end]

11. Pass the currentParameterValues instance to the ApplyCurrentValues method of the ParameterFieldDefinition instance.

[Visual Basic]

```
myParameterFieldDefinition.ApplyCurrentValues(currentParameterValues)
```

[end]

[C#]

```
parameterFieldDefinition.ApplyCurrentValues(currentParameterValues);
```

[end]

This step procedure showed you how to create a method that retrieves submitted values from an ArrayList instance and places them as current values into a ParameterFieldDefinition instance. Now, you must call this method before your report is bound to the CrystalReportViewer control, for the report to be aware that it has parameter settings.

To call the SetCurrentValuesForParameterField() method before the report is bound to the CrystalReportViewer control

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.

   Within these line breaks, you can now enter additional code that modifies the report before it is bound to the viewer.

2. Within the line breaks, declare and instantiate an ArrayList.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

3. Add the city names "Paris" and "Tokyo" as strings to the ArrayList instance.

[Visual Basic]

```
myArrayList.Add("Paris")
myArrayList.Add("Tokyo")
```

[end]

[C#]

```
arrayList.Add("Paris");
arrayList.Add("Tokyo");
```

[end]

4. Call the `SetCurrentValuesForParameterField()` method, and pass in the CustomersByCityReport instance, and the ArrayList instance.

[Visual Basic]

```
SetCurrentValuesForParameterField(customersByCityReport, myArrayList)
```

[end]

[C#]

```
SetCurrentValuesForParameterField(customersByCityReport, arrayList);
```

[end]

The final line of code in the method is code that binds the report to the CrystalReportViewer control.

You are now ready to build and run your project. It is expected that the report displays successfully because there is now code written to set current values into the parameter field.

To test the loading of the CustomersByCity report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The CustomersByCity report displays successfully, showing listings for customers in Paris and Tokyo.

4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you learn how to retrieve the default values from the parameter field and set those values into a ListBox control. These are used at the end of the tutorial to select new cities dynamically and filter the report based on those newly selected cities.

Create a ListBox Control that Displays Default Parameters

The remainder of the tutorial is concerned with displaying a complete list of default values for the parameter field in a ListBox control, and based on selections that you make from that ListBox control, refiltering the contents of the report.

In this section you learn how to populate the ListBox control from the default values of the parameter field.

> **Note** Remember that you set the Default Values, a large list of cities, when you created this report at the beginning of the tutorial.

To do this, you must add and configure a ListBox control, and then create a helper method to populate the ListBox control.

To create and configure a ListBox control on the form

1. Open the Web or Windows form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:

   Click the **CrystalReportViewer** control to select it.

   Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER.

   The CrystalReportViewer control drops by one line.
4. If you are developing a Windows project, do the following:

   Click the **CrystalReportViewer** control to select it.

   From the **Properties** window, set the **Dock** to "Bottom"

   Note In Visual Studio, When you select the **Dock** property, a frame appears instead of a list of options. Select the portion of the frame that corresponds to "Bottom".

   Resize the Windows form and the **CrystalReportViewer** control so that the CrystalReportViewer is large enough to display a report. Leave room above the **CrystalReportViewer** control for a **ListBox** control.

   From the **Properties** window, set the **Anchor** to "Top, Bottom, Left, Right."
5. From the **Toolbox**, drag a **ListBox** control above the **CrystalReportViewer** control.

   Note If a Smart Task appears on the ListBox (when using Visual Studio 2005), press Esc to close it.
6. Click on the **ListBox** control to select it.
7. From the **Properties** window:

   Set the **ID** or **Name** to "defaultParameterValuesList."

   Set the **SelectionMode** to "Multiple" (in a Windows project, "MultiExtended").
8. From the **File** menu, select **Save All**.

You are now ready to create a helper method that retrieves the default values from the parameter field.

To create a helper method that retrieves the default values from the parameter field

1. Open the Web or Windows form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a new private method named `GetDefaultValuesFromParameterField()` that returns an ArrayList instance, with ReportDocument passed into the method signature.

[Visual Basic]

```
Private Function GetDefaultValuesFromParameterField(ByVal
myReportDocument As ReportDocument) As ArrayList


End Function
```

[end]

[C#]

```
private ArrayList GetDefaultValuesFromParameterField(ReportDocument
reportDocument)
{
}
```

[end]

4. Within the `GetDefaultValuesFromParameterField()` method, retrieve the ParameterFieldDefinitions indexed class, which comes from the DataDefinition property of the ReportDocument instance.

[Visual Basic]

```
Dim myParameterFieldDefinitions As ParameterFieldDefinitions =
myReportDocument.DataDefinition.ParameterFields
```

[end]

[C#]

```
ParameterFieldDefinitions parameterFieldDefinitions =
reportDocument.DataDefinition.ParameterFields;
```

[end]

5. Retrieve the ParameterFieldDefinition instance from the ParameterFieldDefinitions indexed class, which is based on the index entry of the PARAMETER_FIELD_NAME constant.

[Visual Basic]

```
Dim myParameterFieldDefinition As ParameterFieldDefinition =
myParameterFieldDefinitions(PARAMETER_FIELD_NAME)
```

[end]

[C#]

```
ParameterFieldDefinition parameterFieldDefinition =
parameterFieldDefinitions[PARAMETER_FIELD_NAME];
```

[end]

6. Retrieve a ParameterValues indexed class (as the variable defaultParameterValues) from the DefaultValues property of the ParameterFieldDefinition instance.

[Visual Basic]

```
Dim defaultParameterValues As ParameterValues =
myParameterFieldDefinition.DefaultValues
```

[end]

[C#]

```
ParameterValues defaultParameterValues =
parameterFieldDefinition.DefaultValues;
```

[end]

7. Declare and instantiate an ArrayList.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

8. Create a foreach loop, to retrieve each ParameterValue instance from defaultParameterValues.

[Visual Basic]

```
For Each myParameterValue As ParameterValue In defaultParameterValues


Next
```

[end]

[C#]

```
foreach(ParameterValue parameterValue in defaultParameterValues)
{

}
```

[end]

Within the foreach loop, you now create a nested conditional block that checks for discrete (as opposed to range) parameter values. Two versions of this conditional block exist, because the API has changed slightly across versions of Crystal Reports for Visual Studio. Check your API (using IntelliSense) to see which property is available under ParameterValue:

9. If the available property is IsRange then, within the foreach loop, enter this code:

[Visual Basic]

```
If (Not myParameterValue.IsRange) Then
```

```
    End If
```
[end]

[C#]

```
    if(!parameterValue.IsRange)
    {
    }
```
[end]

10. Or, if the available property is Kind (DiscreteOrRangeKind, an enum with three values: DiscreteValue, RangeValue, DiscreteAndRangeValue) then, within the foreach loop, enter this code instead:

[Visual Basic]

```
    If (myParameterValue.Kind = DiscreteOrRangeKind.DiscreteValue) Then


    End If
```
[end]

[C#]

```
    if(parameterValue.Kind == DiscreteOrRangeKind.DiscreteValue)
    {
    }
```
[end]

11. Within this nested conditional block, cast the ParameterValue instance to its extended class, DiscreteParameterValue.

[Visual Basic]

```
    Dim myParameterDiscreteValue As ParameterDiscreteValue =
    CType(myParameterValue, ParameterDiscreteValue)
```
[end]

[C#]

```
    ParameterDiscreteValue parameterDiscreteValue =
    (ParameterDiscreteValue)parameterValue;
```
[end]

12. Also within the nested conditional block, add the Value property of the ParameterDiscreteValue instance (converted to String) into the ArrayList instance.

[Visual Basic]

```
    myArrayList.Add(myParameterDiscreteValue.Value.ToString())
```
[end]

[C#]

```
    arrayList.Add(parameterDiscreteValue.Value.ToString());
```
[end]

13. Outside the conditional block, and outside the foreach loop, at the end of the method, return the ArrayList instance from the method.

[Visual Basic]

```
    Return myArrayList
```
[end]

[C#]

```
    return arrayList;
```
[end]

You have retrieved the default values from the parameter field and returned them from the method as an ArrayList. You now bind this ArrayList to the defaultParameterValuesList ListBox control.

Your code varies slightly depending on whether you use a Web project or a Windows project; therefore, be sure to only complete either the Web or Windows procedure below.

To bind the ArrayList returned from the method to the ListBox in a Web project

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code immediately after the line of code that adds the Tokyo string value to ArrayList instance.

   Within these line breaks, you can now enter additional code that sets the data source for the defaultParameterValuesList ListBox control when the page loads for the first time.

2. Within the line breaks, create a `Not IsPostBack` conditional block.

[Visual Basic]

```
    If Not IsPostBack Then


    End If
```
[end]

[C#]

```
    if(!IsPostBack)
    {
    }
```
[end]

   Note The `Not IsPostBack` conditional block is used to encapsulate code that should only be run the first time the page loads. Controls are typically bound to data values within `Not IsPostBack` conditional blocks so that their data values (and any subsequent control events) are not reset during page reloads.

3. Within the `Not IsPostBack` conditional block, set the DataSource property of the defaultParameterValuesList ListBox to the `GetDefaultValuesFromParameterField()` helper method, passing in the CustomersByCity report instance as a method parameter.

[Visual Basic]

```
    defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport)
```
[end]

[C#]

```
    defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport);
```

[end]

4. Still within the `Not IsPostBack` conditional block, call the `DataBind()` method of the defaultParameterValuesList ListBox.

[Visual Basic]

```
    defaultParameterValuesList.DataBind()
```

[end]

[C#]

```
    defaultParameterValuesList.DataBind();
```

[end]

To bind the ArrayList returned from the method to the ListBox in a Windows project

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code immediately after the line of code that adds the Tokyo string value to ArrayList instance.

   Within these line breaks, you can now enter additional code that sets the data source for the defaultParameterValuesList ListBox control when the page loads for the first time..

2. Within the line breaks, set the DataSource property of the defaultParameterValuesList ListBox to the `GetDefaultValuesFromParameterField()` helper method, passing in the CustomersByCity report instance as a method parameter.

[Visual Basic]

```
    defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport)
```

[end]

[C#]

```
    defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport);
```

[end]

You are now ready to build and run the project, to verify whether the defaultParameterValuesList ListBox is populated.

To test the population of the defaultParameterValuesList ListBox control

1. From the **Build** menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The defaultParameterValuesList ListBox control displays a complete list of default values (cities, in our tutorial).

4. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you add a button to redisplay the report based on selections from the defaultParameterValuesList ListBox control.

Setting Parameters from ListBox Selections

In this section you add a button to redisplay the report based on selections from the defaultParameterValuesList ListBox control.

Within the event method for this button, you call the same method that is called when the page first loads: `SetCurrentValuesForParameterField()`. But this time, rather than pass in arbitrary values (Paris and Tokyo) you pass in the selected values from the defaultParameterValuesList ListBox control.

To create and configure a redisplay Button on the form

1. Open the Web or Windows form.
2. From the **View** menu, click **Designer**.
3. From the **Toolbox**, drag a **Button** control to the right of the **ListBox** control.
4. Click on the **Button** control to select it.
5. From the **Properties** window:

   Set the **ID** or **Name** to "redisplay."

   Set the **Text** to "Redisplay Report."

You are now ready to create a button click event method that checks for selected items in the ListBox control and passes them to `SetCurrentValuesForParameterField()` method.

Your code varies slightly for a Web project or a Windows project, so only complete either the Web or Windows procedure below.

To create the click event method for the redisplay Button in a Web project

1. Double-click the redisplay Button control.

   You are taken to the code-behind class where a `redisplay_Click()` event method has been automatically generated.
2. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class for the System.Web.UI.WebControls namespace (if this namespace has not already been declared).

[Visual Basic]

```
Imports System.Web.UI.WebControls
```

[end]

[C#]

```
using System.Web.UI.WebControls;
```

[end]

   Note This declaration is needed, to access the ListItem class.
3. Within the `redisplay_Click()` event method that has just been auto-generated, declare and instantiate an ArrayList.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

4. Create a foreach loop to retrieve each ListItem instance from the Items property of defaultParameterValuesList.

[Visual Basic]

```
For Each item As ListItem In defaultParameterValuesList.Items


Next
```

[end]

[C#]

```
foreach(ListItem item in defaultParameterValuesList.Items)
{
}
```

[end]

5. Within the foreach loop, create a nested conditional block that checks whether the Selected property of the current Item instance is set to True.

[Visual Basic]

```
If item.Selected Then


End If
```

[end]

[C#]

```
if(item.Selected)
{
}
```

[end]

6. Within the conditional block, add the Value property of the Item instance to the ArrayList instance.

[Visual Basic]

```
myArrayList.Add(item.Value)
```

[end]

[C#]

```
arrayList.Add(item.Value);
```

[end]

7. Outside the conditional block, and outside the foreach loop, call the `SetCurrentValuesForParameterField()` method and pass in the CustomersByCity report instance and the ArrayList instance.

[Visual Basic]

```
SetCurrentValuesForParameterField(customersByCityReport, myArrayList)
```

[end]

[C#]

```
SetCurrentValuesForParameterField(customersByCityReport, arrayList);
```
[end]

Now that the selected values from the ListBox control have been applied as the current values for the parameter field, you are ready to redisplay the report.

8. Rebind the CustomersByCity report instance to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]
```
myCrystalReportViewer.ReportSource = customersByCityReport
```
[end]
[C#]
```
crystalReportViewer.ReportSource = customersByCityReport;
```
[end]

To create the click event method for the redisplay Button in a Windows project

1. Double-click the redisplay Button control.

   You are taken to the code-behind class where a `redisplay_Click()` event method has been automatically generated.

2. Within the `redisplay_Click()` event method that has just been auto-generated, declare and instantiate an ArrayList.

[Visual Basic]
```
Dim myArrayList As ArrayList = New ArrayList()
```
[end]
[C#]
```
ArrayList arrayList = new ArrayList();
```
[end]

3. Create a foreach loop, to retrieve each item (as a string) from the SelectedItems property of defaultParameterValuesList.

[Visual Basic]
```
For Each item As String In defaultParameterValuesList.SelectedItems


Next
```
[end]
[C#]
```
foreach(string item in defaultParameterValuesList.SelectedItems)
{
}
```
[end]

4. Within the conditional block, add item String instance to the ArrayList instance.

[Visual Basic]
```
myArrayList.Add(item)
```
[end]
[C#]

```
    arrayList.Add(item);
```

[end]

5. Outside the conditional block, and outside the foreach loop, call the `SetCurrentValuesForParameterField()` method and pass in the CustomersByCity report instance and the ArrayList instance.

[Visual Basic]

```
    SetCurrentValuesForParameterField(customersByCityReport, myArrayList)
```

[end]

[C#]

```
    SetCurrentValuesForParameterField(customersByCityReport, arrayList);
```

[end]

   Now that the selected values from the ListBox control have been applied as the current values for the parameter field, you are ready to redisplay the report.

6. Rebind the CustomersByCity report instance to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
    myCrystalReportViewer.ReportSource = customersByCityReport
```

[end]

[C#]

```
    crystalReportViewer.ReportSource = customersByCityReport;
```

[end]

You are now ready to build and run the project, to verify that the parameter field has been reset successfully.

To test the population of the defaultParameterValuesList ListBox control

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, CTRL-click to select at least four different cities in the list.
5. Click **Redisplay Report**.

   The page reloads and displays the customer records for customers who live in the list of cities that you have selected.

6. Return to Visual Studio and click **Stop** to exit from debug mode.

If you are working with Web Site, continue to Configuring Parameter Persistence.

If the tutorial you are building is Windows based, continue to Conclusion.

Configuring Parameter Persistence

In this section, you configure persistence (in a Web-based tutorial) for the parameter field selections that are made from the ListBox control.

As demonstrated in the tutorial Persisting the ReportDocument Object Model Using Session, changes made within the ReportDocument object model are lost when the Web page is reloaded, each time users click buttons on the CrystalReportViewer toolbar (such as Next Page and Zoom).

To demonstrate lack of persistence for parameter selections

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, SHIFT-click to select all cities in the list.
5. Click **Redisplay Report**.

   The page reloads, and then displays the customer records for all customers in all cities. This is a large report that contains many pages.
6. On the CrystalReportViewer toolbar, click **Next Page**.

   The list of selected cities is not persisted. Page 2 of the report is not displayed. Instead, you see the startup parameter settings again (Paris and Tokyo).

   Note In Visual Studio .NET 2002, the list of selected cities may be persisted due to version differences.
7. Return to Visual Studio and click **Stop** to exit from debug mode.

You must add persistence code to your application so that changes made within the ReportDocument object model are persisted when Web pages are reloaded.

To begin, you add persistence code to the `ConfigureCrystalReports()` method, by adding an Else block to the `Not IsPostBack` conditional block. You then set unique values for the ArrayList instance for either condition in the conditional block. On page startup, you set the default values ("Paris" and "Tokyo") into the ArrayList instance. On page reloads, you retrieve the ArrayList instance that is stored in Session.

To add persistence code to the ConfigureCrystalReports() method

1. In the `ConfigureCrystalReports()` method, cut and paste the two lines of code, which add Paris and Tokyo to the ArrayList, into the bottom of the If `Not IsPostBack` conditional block.

   When you are finished, the conditional block should look like this:

[Visual Basic]

```
    If Not IsPostBack Then
        defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport)
        defaultParameterValuesList.DataBind()
        myArrayList.Add("Paris")
        myArrayList.Add("Tokyo")
    End If
```

[end]

[C#]

```
    if (!IsPostBack)
    {
       defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport);
```

```
        defaultParameterValuesList.DataBind();
        arrayList.Add("Paris");
        arrayList.Add("Tokyo");
    }
```

[end]

2. Add a final line of code to the conditional block that assigns the ArrayList instance to Session.

   Note You can use the variable name as the string identifier for the Session that you add.

[Visual Basic]

```
    Session("myArrayList") = myArrayList
```

[end]

[C#]

```
    Session["arrayList"] = arrayList;
```

[end]

3. Add an Else condition to the `Not IsPostBack` conditional block.
4. Within the Else block, retrieve the ArrayList instance from Session and cast it to ArrayList.

[Visual Basic]

```
    myArrayList = Ctype(Session("myArrayList"), ArrayList)
```

[end]

[C#]

```
    arrayList = (ArrayList)Session["arrayList"];
```

[end]

When you are finished, the conditional block should look like this:

[Visual Basic]

```
    If Not IsPostBack Then
        defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport)
        defaultParameterValuesList.DataBind()
        myArrayList.Add("Paris")
        myArrayList.Add("Tokyo")
        Session("myArrayList") = myArrayList
    Else
        myArrayList = Ctype(Session("myArrayList"), ArrayList)
    End If
```

[end]

[C#]

```
    if (!IsPostBack)
```

```
    {
        defaultParameterValuesList.DataSource =
    GetDefaultValuesFromParameterField(customersByCityReport);
        defaultParameterValuesList.DataBind();
        arrayList.Add("Paris");
        arrayList.Add("Tokyo");
        Session["arrayList"] = arrayList;
    }
    else
    {
        arrayList = (ArrayList)Session["arrayList"];
    }
```

[end]

Those modifications to `ConfigureCrystalReports()` method ensure that the current instance of ArrayList is always available to be passed into the `SetCurrentValuesForParameterField()` method.

In the next section, you make two changes to the code in the button click event:

Take the ArrayList that you created and assign it to Session.

Replace the last two lines of code (that configure and display the report) with a call to the `ConfigureCrystalReports()` method to perform this functionality on a common set of code.

To modify the code in the button click event method to work with Session persistence

1. Delete the following last two lines of code in the button click event method.

   The first line of code to delete is the call to the `SetCurrentValuesForParameterField()` method. The second line of code to delete is the code that binds the customersByCityReport instance to the ReportSource property of the CrystalReportViewer control.

   In the next step, you add two new lines of code to replace the deleted code.

2. Within the button click event method, immediately outside the foreach loop, add a line of code that assigns the ArrayList instance to Session.

   Note You can use the variable name as the string identifier for the Session that you add.

[Visual Basic]

```
    Session("myArrayList") = myArrayList
```

[end]

[C#]

```
    Session["arrayList"] = arrayList;
```

[end]

3. Call the `ConfigureCrystalReports()` method.

This retrieves the ArrayList instance, applies it to the report, and binds the report to the CrystalReportViewer control.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

You are now ready to build and run the project, to verify that the parameter field has been reset successfully.

Note An alternative approach to persistence is to persist the ReportDocument instance. To learn how to persist the ReportDocument instance to Session, see the tutorial Persisting the ReportDocument Object Model Using Session.

To test the population of the defaultParameterValuesList ListBox control

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, SHIFT-click to select all cities in the list.
5. Click **Redisplay Report**.

   The page reloads and displays the customer records for all customers in all cities. This is a large report that contains many pages.
6. On the CrystalReportViewer toolbar, click **Next Page**.
7. The list of selected cities is now persisted. Page 2 of the report is displayed.
8. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully created a report with a discrete parameter value and a helper method that can accept any set of values in a common format (an ArrayList), and then applied those values to a report based on a specific parameter field name (the PARAMETER_FIELD_NAME constant).

You have also learned how to populate a ListBox control with the default values for a particular parameter field. You have learned how to retrieve the selected values from your ListBox and place them into an ArrayList, to recall the original helper method to change the list of parameter values and change the display of the report.

Note You may want to work further with this tutorial and add a subreport into your report, and then configure that subreport with a separate parameter. If so, proceed to the next tutorial, which modifies your current tutorial Reading and Setting Parameters with a Subreport.

To learn about reading and setting discrete parameters with enhanced API features, continue to Addendum: Enhancements to the Discrete Parameters Code.

Addendum: Enhancements to the Discrete Parameters Code

If you have installed Visual Studio 2005 or Crystal Reports Developer you have access to the enhanced API that sets the discrete parameters in the Crystal report. The Crystal Reports Developer API helps minimize the amount of code that is needed to set the discrete parameters.

In the previous procedures, you learned to create the SetCurrentValuesForParameterField() helper method.

In this tutorial, you can delete the helper method and instead call the SetParameterValue() method of the ReportDocument class.

The SetParameterValue() method comes in the following overloaded methods:

SetParameterValue(int index, object value)

SetParameterValue(string parameterFieldName, object value)

SetParameterValue(string parameterFieldName, object value, string subreport)

You can pass in any type of object where the value satisfies the default values for the parameter field. The object can be an Array instance that stores a list of parameter values.

Prerequisites:

You must create a project that is based on the instructions in Reading and Setting Discrete Parameters.

However, in Setting Parameters Manually in Code, you only need to create the PARAMETER_FIELD_NAME constant. You do not need to create the SetCurrentValuesForParameterField() help method.

If you already have a project that is based on the instructions in Reading and Setting Discrete Parameters, delete the SetCurrentValuesForParameterField() helper method and the call within the ConfigureCrystalReports() method.

To use the SetParameterValue() method for Discrete Parameters

1. Open the completed project for this tutorial.
2. Open the Web or Windows Form.
3. From the **View** menu, click **Code**.
4. Within the ConfigureCrystalReports() method, above the line that binds the report to the ReportSource property of the CrystalReportViewer control, call the SetParameterValue() method from the CustomerByCity class. Pass the parameter field name and the parameter values in an Array instance to the method.

[Visual Basic]

```
customersByCityReport.SetParameterValue(PARAMETER_FIELD_NAME,
arrayList.ToArray())
```

[end]

[C#]

```
customersByCityReport.SetParameterValue(PARAMETER_FIELD_NAME,
arrayList.ToArray());
```

[end]

You are now ready to build and run the project, to read and set discrete parameters.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_Parameters

C# Windows project: CS_Win_RDObjMod_Parameters

Visual Basic Web Site: VB_Web_RDObjMod_Parameters

Visual Basic Windows project: VB_Win_RDObjMod_Parameters

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Reading and Setting Parameters with a Subreport

Reading and Setting Parameters with a Subreport

Introduction

In this tutorial, you explore an additional complication: what if the report that requires parameters contains a subreport that requires different parameters?

To do this tutorial, you must complete the previous tutorial, Reading and Setting Discrete Parameters.

In the previous tutorial, Reading and Setting Discrete Parameters, you have learned how to create a report with a parameter, and how to write code to set that parameter at runtime – both with hard-coded parameter values and parameter values that are passed from a ListBox control.

In this tutorial, you learn how to add parameters to a subreport.

You need to make four modifications to the project that you have created in Reading and Setting Discrete Parameters:

- You add a subreport into the original report.

  This subreport addresses the Orders table of the Xtreme database. The Orders table is related to the Customers table that is used in the previous tutorial by a Customer ID foreign key.

- You add a range parameter to the subreport that filters by a range of order dates.

- You add two Text controls to the form: orderStartDate, and orderEndDate, to set the order date range at runtime.

- You add a new method.

  This method creates a ParameterRangeValue instance that contains the startDate and endDate values, and then passes that ParameterRangeValue instance into the range parameter inside the subreport.

When you have completed this tutorial, you can filter the values that are displayed on the report at runtime. The code that you add limits the number of cities that are displayed in the main report, and limits the range of order dates to be displayed in the subreport.

This tutorial can also be completed with classes of the CrystalReportViewer object model, although the ReportDocument object model is preferred.

To build this tutorial using the CrystalReportViewer object model, see Reading and Setting Parameters with a Subreport Using the CrystalReportViewer Object Model.

Adding a Subreport to the Original Report

To begin, you add a subreport to the original report.

To add a subreport

1. Open the project you created in the previous tutorial, Reading and Setting Discrete Parameters.
2. From **Solution Explorer**, double-click the **CustomersByCity** report to open it.
3. Right-click the **Details** gray bar and select **Insert Section Below**.

4. Right-click within the new **Details b** section that you have created, point to **Insert**, and then click **Subreport**.

   A gray square appears around the mouse cursor.
5. Drag the gray rectangle into the new **Details b** section, and then click to release.
6. In the **Insert Subreport** dialog box, on the **Subreport** tab, select **Create a subreport with the Report Wizard**.

   Note The Insert Subreport dialog box includes other options that allow you to choose an existing report and on-demand subreports.
7. In the **New report name** field, type "CustomerOrders."
8. Click **Report Wizard...**
9. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to the full version, the Create New Connection folder does not exist; the contents are shown at the top level.
10. From the subfolder that opens, expand the **ODBC (RDO)** folder.

    The folder contains the database server, which has been configured for the report when the report is created.

    Note If the server is not displayed, follow the instructions in the previous tutorial to connect to the Xtreme Sample Database.
11. Select the **Orders** table and click the **>** symbol to move the Orders table into the **Select Tables** panel, and then click **Next**.
12. From the **Available Fields** panel, select **Order ID**, **Order Date**, **Ship Date**, and **Ship Via**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Finish**.
14. In the **Insert Subreport** dialog box, select the **Link** tab.
15. In the panel **Container Report field(s) to link to**, in the list **Available fields**, expand the **Customers** table, select **Customer ID**, and then click the **>** symbol.
16. In the **Customers.Customer ID field link** panel that appears, leave the default selections unchanged.

    These parameter and data selections auto-generate a relationship between the main report and the subreport.
17. Click **OK**.

    The new subreport, CustomerOrders, is displayed within the Details b section of the Main report.

    Note When you add a subreport to the Details section, the subreport displays for every row, which adds a performance cost to your report. If you do not need subreport information with that level of granularity, place the subreport in a Group section rather than a Details section.

You are now ready to verify the settings in the subreport.

To verify the settings in the subreport

1. In the report Details section, double-click on the **CustomerOrders** subreport to view it.

   At the bottom of the designer view, navigation buttons appear for both the Main Report and the CustomerOrders subreport.
2. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click **Toggle Field View**.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.
3. In the **Field Explorer**, expand **Parameter Fields**.
4. Verify that the parameter field **Pm-Customers.Customer ID** was auto-generated when the subreport was linked.
5. On the toolbar, click **Select Expert**.
6. In the **Select Expert** dialog box, verify that the criteria **Orders.Customer ID is equal to {Pm-Customers.Customer ID}** is set, and then click **OK**.
7. From the **File** menu, select **Save All**.

   You have successfully added a CustomerOrders subreport to the CustomersByCity report. In the next section, you add an OrderDateRange parameter to the subreport.

To add an OrderDateRange parameter to the subreport

1. In the **Field Explorer**, right-click **Parameter Fields** and select **New...**
2. In the Create Parameter Field dialog box:

   Set the **Name** to "OrderDateRange."

   Set the **Prompting text** to "Specify a Date Range of Orders to display."

   Set the **Value type** to "**Date**."

   Set the **Options** to one selection only, "**Range value(s)**."
3. Click **OK**.
4. On the toolbar, click **Select Expert**.
5. Click the **New** tab.
6. In the **Choose Field** dialog box, expand the **Orders** table, select **Order Date**, and then click **OK**.
7. On the new **Orders.OrderDate** tab, from the drop down criteria list, select **formula:**
8. Type the following formula:

```
{Orders.Order Date} in {?OrderDateRange}
```

9. Click **OK**.
10. From the **File** menu, select **Save All**.

    You have successfully added an OrderDateRange parameter to the subreport and linked it to the Orders.OrderDate column. In the next section, you add code to address the OrderDateRange parameter within the subreport.

Adding the Subreport Parameter Code

You are now ready to add the parameter code for the subreport to the code-behind class. To begin, you create a private helper method, SetDateRangeForOrders().

To create and code the SetDateRangeForOrders() method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the top of the class, add two new constants below the existing PARAMETER_FIELD_NAME constant added during the previous tutorial.

[Visual Basic]

```
Private Const SUBREPORT_PARAMETER_FIELD_NAME As String = "OrderDateRange"
Private Const SUBREPORT_NAME As String = "CustomerOrders"
```

[end]

[C#]

```
private const string SUBREPORT_PARAMETER_FIELD_NAME = "OrderDateRange";
private const string SUBREPORT_NAME = "CustomerOrders";
```

[end]

4. At the bottom of the class, create a new private method named `SetDateRangeForOrders()` with three parameters: ReportDocument, a string startDate, and a string endDate.

[Visual Basic]

```
Private Sub SetDateRangeForOrders(ByVal myReportDocument As
ReportDocument, ByVal startDate As String, ByVal endDate As String)


End Sub
```

[end]

[C#]

```
private void SetDateRangeForOrders(ReportDocument reportDocument, string
startDate, string endDate)
{
}
```

[end]

5. Within this method, declare and instantiate the ParameterRangeValue class.

[Visual Basic]

```
Dim myParameterRangeValue As ParameterRangeValue = New
ParameterRangeValue()
```

[end]

[C#]

```
ParameterRangeValue parameterRangeValue = new ParameterRangeValue();
```

[end]

> **Note** For the ParameterRangeValue class to be accessible, you must include an `"Imports" [Visual Basic]` or `"using" [C#]` statement at the top of the code-behind class for the CrystalDecisions.Shared namespace. (You added this declaration in Appendix: Project Setup.)

6. Set the StartValue property of the ParameterRangeValue instance to the startDate method parameter.

[Visual Basic]

```
myParameterRangeValue.StartValue = startDate
```

[end]

[C#]

```
parameterRangeValue.StartValue = startDate;
```

[end]

Note The StartValue and EndValue properties of the ParameterRangeValue class accept values of type Object. This generic type allows the range value that is passed in to be of many types, including: text, number, date, currency, or time.

7. Set the EndValue property of the ParameterRangeValue instance to the endDate method parameter.

[Visual Basic]

```
myParameterRangeValue.EndValue = endDate
```

[end]

[C#]

```
parameterRangeValue.EndValue = endDate;
```

[end]

8. Set the lower and upper boundaries to be bound-inclusive.

[Visual Basic]

```
myParameterRangeValue.LowerBoundType = RangeBoundType.BoundInclusive
myParameterRangeValue.UpperBoundType = RangeBoundType.BoundInclusive
```

[end]

[C#]

```
parameterRangeValue.LowerBoundType = RangeBoundType.BoundInclusive;
parameterRangeValue.UpperBoundType = RangeBoundType.BoundInclusive;
```

[end]

Note For BoundInclusive, the upper and lower range values are included in the range.

You are now ready to assign the ParameterRangeValue instance to the parameter of the subreport.

9. Retrieve the ParameterFieldDefinitions class, which comes from the DataDefinition property of the ReportDocument instance

Note ParameterFieldDefinitions is an indexed class that contains instances of the ParameterFieldDefinition class.

[Visual Basic]

```
Dim myParameterFieldDefinitions As ParameterFieldDefinitions =
myReportDocument.DataDefinition.ParameterFields
```

[end]

[C#]

```
ParameterFieldDefinitions parameterFieldDefinitions =
reportDocument.DataDefinition.ParameterFields;
```

[end]

10. Retrieve the ParameterFieldDefinition instance from the ParameterFieldDefinitions indexed class, which is based on two indexed values: the subreport parameter field name and the subreport name. Pass in the two constant values that you declared at the top of the class.

[Visual Basic]

```
Dim myParameterFieldDefinition As ParameterFieldDefinition =
myParameterFieldDefinitions(SUBREPORT_PARAMETER_FIELD_NAME,
SUBREPORT_NAME)
```

[end]

[C#]

```
ParameterFieldDefinition parameterFieldDefinition =
parameterFieldDefinitions[SUBREPORT_PARAMETER_FIELD_NAME,
SUBREPORT_NAME];
```

[end]

11. Call the `Clear()` method of the CurrentValues property of the ParameterFieldDefinition instance to remove any existing values from the CurrentValues property.

[Visual Basic]

```
myParameterFieldDefinition.CurrentValues.Clear()
```

[end]

[C#]

```
parameterFieldDefinition.CurrentValues.Clear();
```

[end]

12. Add the ParameterRangeValue instance, which you created earlier, to the CurrentValues property of the ParameterFieldDefinition instance.

[Visual Basic]

```
myParameterFieldDefinition.CurrentValues.Add(myParameterRangeValue)
```

[end]

[C#]

```
parameterFieldDefinition.CurrentValues.Add(parameterRangeValue);
```

[end]

13. Call the `ApplyCurrentValues()` method, passing into it the CurrentValues property of the ParameterFieldDefinition instance.

[Visual Basic]

```
myParameterFieldDefinition.ApplyCurrentValues(myParameterFieldDefinitio
n.CurrentValues)
```

[end]

[C#]

```
parameterFieldDefinition.ApplyCurrentValues(parameterFieldDefinition.Cu
rrentValues);
```

[end]

This step procedure has set start and end date values into a ParameterRangeValue instance and passed those values to the OrderDateRange parameter in the CustomerOrders subreport.

Adding TextBox Controls to Hold Range Parameter Values

In this section, you add two TextBox controls to provide start and end date values at runtime to the OrderDateRange range parameter in the CustomerOrders subreport.

Note If you implement this tutorial in a Web Site, the persistence of date values that users enter into the text boxes are maintained by ViewState.

To create and configure a redisplay Button on the form

1. Open the Web or Windows form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:
   a) Click between the **ListBox** control and the **Button** control.
   b) Press ENTER three times to create two rows between the ListBox control and the Button control.
   c) In the first row created below the **ListBox** control, type "Order Start Date."
   d) In the second row created below the **ListBox** control, type "Order End Date."
4. If you are developing a Windows project, do the following:
   a) From the **Toolbox**, drag two **Label** controls to the right of the ListBox control. Place one label above the other, with both of them above the **Button** control.
   b) Select the first **Label** control. From the **Properties** window, set the **Text** property to "Order Start Date."
   c) Select the second **Label** control. From the **Properties** window, set the **Text** property to "Order End Date."

   The remaining steps apply to both Web and Windows projects.
5. From the **Toolbox,** drag a **TextBox** control to the right of "Order Start Date."
6. Click on the **TextBox** control to select it.
7. From the **Properties** window, set the **ID** (or **Name**) to "orderStartDate."
8. From the **Toolbox,** drag a **TextBox** control to the right of "Order End Date."
9. Click on the **TextBox** control to select it.
10. From the **Properties** window, set the **ID** (or **Name**) to "orderEndDate."
11. From the **File** menu, select **Save All**.

Modifying Methods to Call the Subreport

You must now modify the `ConfigureCrystalReports()` method and the `redisplay_Click()` event method to receive information from these TextBox controls

and apply them to the `SetDateRangeForOrders()` method, to have the parameter information processed for subreports.

In the previous tutorial, Reading and Setting Discrete Parameters, you designed these methods in two different ways, depending on whether you included Session persistence.

Note Windows projects do not require Session persistence. Web Sites typically require Session persistence.

Choose from one (but not both) of the step procedures below. Either modify the methods that exclude Session persistence, or modify the methods that include Session persistence:

Modifying Methods that Exclude Session Persistence.

Modifying Methods that Include Session Persistence.

Modifying Methods that Exclude Session Persistence

If you created the previous tutorial Reading and Setting Discrete Parameters and excluded Session persistence, work through the following procedures. If you want to include Session persistence, see Modifying the Methods that Include Session Persistence.

To modify the ConfigureCrystalReports() method that excludes Session persistence

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the lines which assign "Paris" and "Tokyo" as ArrayList variables.
2. Within the line breaks, declare and set hard-coded values for two string variables, startDate and endDate.

[Visual Basic]

```
Dim startDate As String = "8/1/1997"
Dim endDate As String = "8/31/1997"
```

[end]

[C#]

```
string startDate = "8/1/1997";
string endDate = "8/31/1997";
```

[end]

3. Create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
4. Within the line breaks, enter a call to the `SetDateRangeForOrders()` method and pass in the CustomersByCity report and the startDate and endDate variables.

[Visual Basic]

```
SetDateRangeForOrders(customersByCityReport, startDate, endDate)
```

[end]

[C#]

```
SetDateRangeForOrders(customersByCityReport, startDate, endDate);
```

[end]

This is followed by the original code that binds the report to the CrystalReportViewer control.

5. From the **File** menu, select **Save All**.

Next, you modify the redisplay_Click event method.

To modify the redisplay_Click() method that excludes Session persistence

1. In the `redisplay_Click()` event method, create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
2. Within the line breaks, declare and set values for two string variables, startDate and endDate, from the TextBox controls that you added to the Web or Windows form.

[Visual Basic]

```
Dim startDate As String = orderStartDate.Text
Dim endDate As String = orderEndDate.Text
```

[end]

[C#]

```
string startDate = orderStartDate.Text;
string endDate = orderEndDate.Text;
```

[end]

3. Enter a call to the `SetDateRangeForOrders()` method and pass in the CustomersByCity report and the startDate and endDate variables.

[Visual Basic]

```
SetDateRangeForOrders(customersByCityReport, startDate, endDate)
```

[end]

[C#]

```
SetDateRangeForOrders(customersByCityReport, startDate, endDate);
```

[end]

4. From the **File** menu, select **Save All**.

This is followed by the original code that binds the report to the CrystalReportViewer control.

Modifying Methods that Include Session Persistence

If you created the previous tutorial Reading and Setting Discrete Parameters and included Session persistence, work through the following procedures. Otherwise, continue to Modifying the Methods that Exclude Session Persistence.

To modify the ConfigureCrystalReports() method that includes Session persistence

1. In the `ConfigureCrystalReports()` method, create a couple of line breaks in the code after the line that declares and instantiates ArrayList.
2. Within the line breaks, declare two string variables, startDate and endDate.

[Visual Basic]

```
Dim startDate As String
Dim endDate As String
```

[end]

[C#]

```
string startDate;
string endDate;
```

[end]

3. Within the `Not IsPostBack` conditional block, enter default values for the startDate and endDate variables.

[Visual Basic]

```
startDate = "8/1/1997"
endDate = "8/31/1997"
```

[end]

[C#]

```
startDate = "8/1/1997";
endDate = "8/31/1997";
```

[end]

4. Within the `Not IsPostBack` conditional block, assign the startDate and endDate variables into Session.

[Visual Basic]

```
Session("startDate") = startDate
Session("endDate") = endDate
```

[end]

[C#]

```
Session["startDate"] = startDate;
Session["endDate"] = endDate;
```

[end]

5. Within the Else block, after the ArrayList instance is retrieved from Session, retrieve the startDate and endDate variables from Session.

[Visual Basic]

```
startDate = Session("startDate").ToString()
endDate = Session("endDate").ToString()
```

[end]

[C#]

```
startDate = Session["startDate"].ToString();
endDate = Session["endDate"].ToString();
```

[end]

   With that approach, you reach the end of the block with the date variables assigned in either case. This follows parallel logic to the assignment of the ArrayList variable that you configured in the previous tutorial.

6. Create a couple of line breaks in the code above the line that binds the report to the CrystalReportViewer control.
7. Within these new line breaks, enter a call to the `SetDateRangeForOrders()` method and pass in the CustomersByCity report and the startDate and endDate variables.

[Visual Basic]

```
SetDateRangeForOrders(customersByCityReport, startDate, endDate)
```

[end]

[C#]

```
SetDateRangeForOrders(customersByCityReport, startDate, endDate);
```

[end]

This is followed by the original code that binds the report to the CrystalReportViewer control.

8. From the **File** menu, select **Save All**.

Next, you modify the redisplay_Click event method.

To modify the redisplay_Click() method that includes Session persistence

1. In the `redisplay_Click()` event method, create a couple of line breaks in the code after the line that assigns the ArrayList instance to Session.
2. Within the line breaks, assign the Text property of the orderStartDate TextBox and the orderEndDate TextBox to Session variables.

[Visual Basic]

```
Session("startDate") = orderStartDate.Text
Session("endDate") = orderEndDate.Text
```

[end]

[C#]

```
Session["startDate"] = orderStartDate.Text;
Session["endDate"] = orderEndDate.Text;
```

[end]

3. From the **File** menu, select **Save All**.

Those startDate and endDate Session values are now retrieved and applied when the `ConfigureCrystalReports()` method is called.

You are now ready to build and run the project, to verify that the TextBox values are resetting the range parameter in the subreport.

Testing the Setting of the Subreport Parameter

You are now ready to test the setting of the subreport parameter from the TextBox values.

To test the setting of the subreport parameter

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. In the **ListBox** control, CTRL-click to select at least four different cities in the list.
5. In the startDate **TextBox** control, enter "1/1/1997."
6. In the endDate **TextBox** control, enter "12/31/1997."
7. Click the **Redisplay Report** button.

The page reloads and displays the customer records for customers who live in the list of cities that have just been selected, as well as a subreport that displays orders for the date range specified above.

8. In the **CrystalReportViewer** control, increase the **Zoom** level to 125%.

   The page reloads at 125% zoom. The values that are selected for both cities and order date range are persisted.

9. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

You have successfully modified your tutorial project to use a report containing a subreport, and set an order date range to the range parameter that is created in the subreport.

To learn about reading and setting parameters in a subreport with enhanced API features, continue to Addendum: Enhancements to the Range Parameters Code for Subreports.

Addendum: Enhancements to the Range Parameters Code for Subreports

If you have installed Visual Studio 2005 or Crystal Reports Developer you have access to the enhanced API that sets the range parameters in the Crystal report.

In the previous procedures, you learned how to create the SetDateRangeForOrders() helper method that uses the ParameterFieldDefinitions and ParameterFieldDefinition classes.

In this tutorial, you must remove the lines of code that uses the ParameterFieldDefinitions and ParameterFieldDefinition classes. Then, you learn how to use the ParameterFields and ParameterField classes of the enhanced Crystal Reports Developer API to code the SetDateRangeForOrders() method.

Note The enhanced API includes the SetParameterValue(string parameterFieldName, object value, string subreport) method for subreports with discrete parameter fields. Therefore, SetParameterValue() cannot be used in this tutorial because the subreport has a range parameter.

Prerequisites:

You must create a project based on the instructions in Reading and Setting Parameters with a Subreport.

To use the enhanced Crystal Reports API for Subreports with Range Parameters

1. Open the completed project for this tutorial.
2. Open the Web or Windows Form.
3. From the **View** menu, click **Code**.
4. Within the `SetDateRangeForOrders()` method, delete the lines of code that use the ParameterFieldDefinitions or ParameterFieldDefinition classes. Delete the following lines of code:

[Visual Basic]

```
Dim myParameterFieldDefinitions As ParameterFieldDefinitions =
myReportDocument.DataDefinition.ParameterFields
```

```
Dim myParameterFieldDefinition As ParameterFieldDefinition =
myParameterFieldDefinitions(SUBREPORT_PARAMETER_FIELD_NAME,
SUBREPORT_NAME)
myParameterFieldDefinition.CurrentValues.Clear()
myParameterFieldDefinition.CurrentValues.Add(myParameterRangeValue)
myParameterFieldDefinition.ApplyCurrentValues(myParameterFieldDefinitio
n.CurrentValues)
```

[end]

[C#]

```
ParameterFieldDefinitions parameterFieldDefinitions =
reportDocument.DataDefinition.ParameterFields;
ParameterFieldDefinition parameterFieldDefinition =
parameterFieldDefinitions[SUBREPORT_PARAMETER_FIELD_NAME,
SUBREPORT_NAME];
parameterFieldDefinition.CurrentValues.Clear();
parameterFieldDefinition.CurrentValues.Add(parameterRangeValue);
parameterFieldDefinition.ApplyCurrentValues(parameterFieldDefinition.Cu
rrentValues);
```

[end]

In the next step, you add the new Crystal Reports API methods to the SetDateRangeForOrders() method, after the code that sets the range bound type.

5. Within the `SetDateRangeForOrders()` method, retrieve the ParameterFields instance from the ParameterFields property of the ReportDocument instance.

[Visual Basic]

```
Dim myParameterFields As ParameterFields = reportDocument.ParameterFields
```

[end]

[C#]

```
ParameterFields parameterFields = reportDocument.ParameterFields;
```

[end]

6. Retrieve the ParameterField instance from the ParameterFields indexed class, which is based on two indexed values: the subreport parameter field name and the subreport name. Pass in the two constant values that you declared at the top of the class.

[Visual Basic]

```
Dim myParameterField As ParameterField =
myParameterFields(SUBREPORT_PARAMETER_FIELD_NAME, SUBREPORT_NAME)
```

[end]

[C#]

```
ParameterField parameterField =
parameterFields[SUBREPORT_PARAMETER_FIELD_NAME, SUBREPORT_NAME];
```

[end]

7. Call the `Clear()` method of the CurrentValues property of the ParameterField instance to remove any existing values from the CurrentValues property.

[Visual Basic]

```
myParameterField.CurrentValues.Clear()
```

[end]

[C#]

```
parameterField.CurrentValues.Clear();
```

[end]

8. Add the ParameterRangeValue instance, which you created earlier, to the CurrentValues property of the ParameterField instance.

[Visual Basic]

```
myParameterField.CurrentValues.Add(myParameterRangeValue)
```

[end]

[C#]

```
parameterField.CurrentValues.Add(parameterRangeValue);
```

[end]

You are now ready to build and run the project, to read and set range parameters for a subreport.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_ParametersSubrpt

C# Windows project: CS_Win_RDObjMod_ParametersSubrpt

Visual Basic Web Site: VB_Web_RDObjMod_ParametersSubrpt

Visual Basic Windows project: VB_Win_RDObjMod_ParametersSubrpt

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Exporting to Multiple Formats

Exporting to Multiple Formats

Introduction

In this tutorial, you learn how to export the report programmatically.

Crystal Reports can export reports to the following formats:

- Adobe Acrobat (.pdf)
- Crystal Reports (.rpt)
- Rich Text Format (.rtf)
- Microsoft Word (.doc)
- Microsoft Excel (.xls)
- HTML 3.2 (.htm)
- HTML 4.0 (.htm)

You can select any of those formats, click the Export button for the CrystalReportViewer control, and export a report in the selected format to your local machine that runs Crystal Reports in a Web or Windows application.

You can also export reports programmatically, to specific directories on the local Web server or Windows machine.

To begin this tutorial, you add a DropDownList control to your Web or Windows Form, and then populate it with the values from the ExportFormatType enum in the CrystalDecisions.Shared namespace.

Then, you create three private helper methods that contain the export functionality and that perform specific configuration for each of the export formats.

Finally, you create a click event method from a Button control on the Web or Windows Form, and call the three private helper methods that perform the export.

Adding Controls to the Web or Windows Form

In this section, you add a DropDownList, Button, and Label control above the CrystalReportViewer control on the Web or Windows Form.

To add controls to the Web or Windows Form

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   b) Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER.

      The CrystalReportViewer control drops by one line.

4. If you are developing a Windows project, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   b) From the **Properties** window, set **Dock** to "Bottom."
   c) Resize the **CrystalReportViewer** control, so that you leave enough room above it for a ComboBox control.
   d) From the **Properties** window, set **Anchor** to "Top, Bottom, Left, Right."
5. From the **Toolbox**, drag a **DropDownList** control (in Web Sites) or **ComboBox** control (in Windows projects) above the **CrystalReportViewer** control.

   Note If a Smart Task appears on the DropDownList (ComboBox) when you use Visual Studio 2005, press Esc to close it.
6. Click the **DropDownList** (ComboBox) control to select it.
7. From the **Properties** window, set the **ID** property to "exportTypesList."
8. From the **Toolbox**, drag a **Button** control to the right of the **DropDownList** (ComboBox) control.
9. Click the **Button** control to select it.
10. From the **Properties** window, do the following:
    Set the **ID** property to "exportByType."
    Set the **Text** property to "Export As Selected Type."
11. From the **Toolbox**, drag a **Label** control to the right of the **Button** control.
12. Click the **Label** control to select it.
13. From the **Properties** window, do the following:
    Set the **ID** property to "message."
    Set the **Text** property to be blank.
    Set the **Visible** property to "False."
14. From the **File** menu, select **Save All**.

Now you must populate the DropDownList control from the ExportFormatType enum of the CrystalDecisions.Shared namespace.

To populate the DropDownList control from the ExportFormatType enum for a Web Site

1. Open the Web Form.
2. From the **View** menu, click **Code**.
3. Within the `ConfigureCrystalReports()` method, at the bottom of the method, add a `Not IsPostBack` conditional block.

[Visual Basic]

```
    If Not IsPostBack Then


    End If
```

[end]

[C#]

```
    if (!IsPostBack)
```

```
    {

    }
```
[end]

4. Within the conditional block, set the DataSource property of the exportTypesList ComboBox control to the values of ExportFormatType enum.

[Visual Basic]
```
    exportTypesList.DataSource =
    System.Enum.GetValues(GetType(ExportFormatType))
```
[end]
[C#]
```
    exportTypesList.DataSource =
    System.Enum.GetValues(typeof(ExportFormatType));
```
[end]

5. Call the `DataBind()` method of the exportTypesList DropDownList control to bind the values to the control.

[Visual Basic]
```
    exportTypesList.DataBind()
```
[end]
[C#]
```
    exportTypesList.DataBind();
```
[end]

To populate the DropDownList control from the ExportFormatType enum for a Windows project

1. Open the Windows Form.
2. From the **View** menu, click **Code**.
3. Within the `ConfigureCrystalReports()` method, at the bottom of the method, set the DataSource property of the exportTypesList ComboBox control to the values of the ExportFormatType enum.

[Visual Basic]
```
    exportTypesList.DataSource =
    System.Enum.GetValues(GetType(ExportFormatType))
```
[end]
[C#]
```
    exportTypesList.DataSource =
    System.Enum.GetValues(typeof(ExportFormatType));
```
[end]

Creating Three Methods That Perform the Export

In this section, you create the three private helper methods that perform the export.

```
ExportSetup()
ExportSelection()
ExportCompletion()
```

These methods are called later in this tutorial, from a button click event method. To begin, you create the `ExportSetup()` helper method.

To create the ExportSetup() method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the top of the class, add three class declarations.

[Visual Basic]

```
Private exportPath As String
Private myDiskFileDestinationOptions As DiskFileDestinationOptions
Private myExportOptions As ExportOptions
```

[end]

[C#]

```
private string exportPath;
private DiskFileDestinationOptions diskFileDestinationOptions;
private ExportOptions exportOptions;
```

[end]

   You later instantiate those helper classes in the `ExportSetup()` method.

4. At the bottom of the class, create a private helper method named `ExportSetup()` with no return value.

[Visual Basic]

```
Public Sub ExportSetup()


End Sub
```

[end]

[C#]

```
private void ExportSetup()
{
}
```

[end]

5. Within the method, set the exportPath string variable to the root directory of the hard drive.

[Visual Basic]

```
exportPath = "C:\Exported\"
```

[end]

[C#]

```
exportPath = "C:\\Exported\\";
```

[end]

Note If you want to place the Exported folder within the Web directory of your Web server, prefix the folder name with the Request.PhysicalApplicationPath property.

6. Create a conditional block that tests whether the directory in the exportPath string already exists.

[Visual Basic]

```
If Not System.IO.Directory.Exists(exportPath) Then


End If
```

[end]

[C#]

```
if (!System.IO.Directory.Exists(exportPath))
{
}
```

[end]

7. Within the conditional block, call the `CreateDirectory()` method of System.IO.Directory to create the directory in the exportPath string.

[Visual Basic]

```
System.IO.Directory.CreateDirectory(exportPath)
```

[end]

[C#]

```
System.IO.Directory.CreateDirectory(exportPath);
```

[end]

8. Outside the conditional block, instantiate the DiskFileDesintationOptions class.

[Visual Basic]

```
myDiskFileDestinationOptions = New DiskFileDestinationOptions()
```

[end]

[C#]

```
diskFileDestinationOptions = new DiskFileDestinationOptions();
```

[end]

9. Populate the ExportOptions instance with the ExportOptions property of the hierarchicalGroupingReport instance.

[Visual Basic]

```
myExportOptions = hierarchicalGroupingReport.ExportOptions
```

[end]

[C#]

```
exportOptions = hierarchicalGroupingReport.ExportOptions;
```

[end]

Note You have instantiated and bound the hierarchicalGroupingReport instance to the CrystalReportViewer control in Additional Setup Requirements of Appendix: Project Setup.

10. Set the ExportDestinationType property of the ExportOptions instance to the enum selection ExportDestinationType.DiskFile.

[Visual Basic]

```
myExportOptions.ExportDestinationType = ExportDestinationType.DiskFile
```

[end]

[C#]

```
exportOptions.ExportDestinationType = ExportDestinationType.DiskFile;
```

[end]

11. For a Windows project, clear the values in the FormatOptions property of the ExportOptions instance. (That line of code is not needed for a Web Site, because the variable is automatically cleared on each click event.)

[Visual Basic]

```
myExportOptions.FormatOptions = Nothing
```

[end]

[C#]

```
exportOptions.FormatOptions = null;
```

[end]

You now create the ExportSelection() helper method.

To create the ExportSelection() method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the top of the class, add a Boolean declaration that is used to test if no export format is selected.

[Visual Basic]

```
Private selectedNoFormat As Boolean = False
```

[end]

[C#]

```
private bool selectedNoFormat = false;
```

[end]

4. At the bottom of the class, create a private helper method named `ExportSelection()` with no return value.

[Visual Basic]

```
Public Sub ExportSelection()


End Sub
```

[end]

[C#]

```
private void ExportSelection()
```

```
    {

    }
```

[end]

5. Within the method, create a "Select Case" [Visual Basic] or "switch" [C#] statement that references the members of the ExportFormatType enum. The enum is based on the SelectedIndex of the exportTypesList DropDownList control that you created in the previous procedure.

[Visual Basic]

```
    Select Case exportTypesList.SelectedIndex

        Case ExportFormatType.NoFormat

        Case ExportFormatType.CrystalReport

        Case ExportFormatType.RichText

        Case ExportFormatType.WordForWindows

        Case ExportFormatType.Excel

        Case ExportFormatType.PortableDocFormat

        Case ExportFormatType.HTML32

        Case ExportFormatType.HTML40

    End Select
```

[end]

[C#]

```
    switch ((ExportFormatType)exportTypesList.SelectedIndex)
    {
       case ExportFormatType.NoFormat:
             break;
       case ExportFormatType.CrystalReport:
             break;
       case ExportFormatType.RichText:
             break;
```

```
        case ExportFormatType.WordForWindows:
              break;
        case ExportFormatType.Excel:
              break;
        case ExportFormatType.PortableDocFormat:
              break;
        case ExportFormatType.HTML32:
              break;
        case ExportFormatType.HTML40:
              break;
    }
```

[end]

You now create the `ExportCompletion()` helper method.

To create the ExportCompletion() method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ExportCompletion()` with no return value.

[Visual Basic]

```
    Public Sub ExportCompletion()


    End Sub
```

[end]

[C#]

```
    private void ExportCompletion()
    {
    }
```

[end]

4. Within the method, create a try/catch block with the Exception class that is referenced as a variable named "ex."

[Visual Basic]

```
    Try


    Catch ex As Exception


    End Try
```

[end]

[C#]

```
try
{
}
catch (Exception ex)
{
}
```

[end]

5. Within the try block, create a conditional block to test the Boolean variable, selectedNoFormat.

[Visual Basic]

```
If selectedNoFormat Then


Else


End If
```

[end]

[C#]

```
if (selectedNoFormat)
{
}
else
{
}
```

[end]

6. Within the If block, set the Text property of the message Label control to the FORMAT_NOT_SUPPORTED constant of the MessageConstants class.

   Note You created the MessageConstants class for this tutorial in Additional Setup Requirements of Appendix: Project Setup.

[Visual Basic]

```
message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
```

[end]

[C#]

```
message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
```

[end]

7. Within the Else block, call the `Export()` method of the hierarchicalGroupingReport instance.

[Visual Basic]

```
hierarchicalGroupingReport.Export()
```

[end]

[C#]

```
hierarchicalGroupingReport.Export();
```

[end]

8. Still within the Else block, set the Text property of the message Label control to the SUCCESS constant of the MessageConstants class.

[Visual Basic]

```
message.Text = MessageConstants.SUCCESS
```

[end]

[C#]

```
message.Text = MessageConstants.SUCCESS;
```

[end]

9. Within the catch block, set the Text property of the message Label control to the FAILURE constant of the MessagesConstants class, and then append to it the Message property of the Exception parameter.

[Visual Basic]

```
message.Text = MessageConstants.FAILURE & ex.Message
```

[end]

[C#]

```
message.Text = MessageConstants.FAILURE + ex.Message;
```

[end]

10. Outside the try/catch block, set the Visible property of the message Label control to "True."

[Visual Basic]

```
message.Visible = True
```

[end]

[C#]

```
message.Visible = true;
```

[end]

11. For a Windows project, reset the selectedNoFormat boolean variable to false. (That line of code is not needed for a Web Site, because the variable is automatically reset to False on each click event.)

[Visual Basic]

```
selectedNoFormat = False
```

[end]

[C#]

```
selectedNoFormat = false;
```

[end]

You have finished creating the three private helper methods that perform the export.

Creating Methods That Configure Multiple Export Formats

In this section, you create the private helper methods that configure the multiple export formats. All of these method are used similarly, except for ConfigureExportToHtml32() and ConfigureExportToHtml40(), which offer different ways to export to the HTML format.

```
ConfigureExportToRpt()
ConfigureExportToRtf()
ConfigureExportToDoc()
ConfigureExportToXls()
ConfigureExportToPdf()
ConfigureExportToHtml32()
ConfigureExportToHtml40()
```

To create the ConfigureExportToRpt() helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToRpt()` with no return value.

[Visual Basic]

```
Public Sub ConfigureExportToRpt()


End Sub
```

[end]

[C#]

```
private void ConfigureExportToRpt()
{
}
```

[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection CrystalReport.

[Visual Basic]

```
myExportOptions.ExportFormatType = ExportFormatType.CrystalReport
```

[end]

[C#]

```
exportOptions.ExportFormatType = ExportFormatType.CrystalReport;
```

[end]

5. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has an .rpt file extension.

[Visual Basic]

```
   myDiskFileDestinationOptions.DiskFileName = exportPath & "Report.rpt"
```
[end]
[C#]
```
   diskFileDestinationOptions.DiskFileName = exportPath + "Report.rpt";
```
[end]

6. Finally, set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you have configured in the previous step.

[Visual Basic]
```
   myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```
[end]
[C#]
```
   exportOptions.DestinationOptions = diskFileDestinationOptions;
```
[end]

To create the ConfigureExportToRtf helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToRtf()` with no return value.

[Visual Basic]
```
   Public Sub ConfigureExportToRtf()


   End Sub
```
[end]
[C#]
```
   private void ConfigureExportToRtf()
   {
   }
```
[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection RichText.

[Visual Basic]
```
   myExportOptions.ExportFormatType = ExportFormatType.RichText
```
[end]
[C#]
```
   exportOptions.ExportFormatType = ExportFormatType.RichText;
```
[end]

5. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has an .rtf file extension.

[Visual Basic]

```
myDiskFileDestinationOptions.DiskFileName = exportPath &
"RichTextFormat.rtf"
```

[end]

[C#]

```
diskFileDestinationOptions.DiskFileName = exportPath +
"RichTextFormat.rtf";
```

[end]

6. Finally, set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you have configured in the previous step.

[Visual Basic]

```
myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```

[end]

[C#]

```
exportOptions.DestinationOptions = diskFileDestinationOptions;
```

[end]

To create the ConfigureExportToDoc helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToDoc()` with no return value.

[Visual Basic]

```
Public Sub ConfigureExportToDoc()


End Sub
```

[end]

[C#]

```
private void ConfigureExportToDoc()
{
}
```

[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection WordForWindows.

[Visual Basic]

```
myExportOptions.ExportFormatType = ExportFormatType.WordForWindows
```

[end]

[C#]

```
exportOptions.ExportFormatType = ExportFormatType.WordForWindows;
```

[end]

5. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has a .doc file extension.

[Visual Basic]

```
myDiskFileDestinationOptions.DiskFileName = exportPath & "Word.doc"
```

[end]

[C#]

```
diskFileDestinationOptions.DiskFileName = exportPath + "Word.doc";
```

[end]

6. Finally, set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you have configured in the previous step.

[Visual Basic]

```
myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```

[end]

[C#]

```
exportOptions.DestinationOptions = diskFileDestinationOptions;
```

[end]

To create the ConfigureExportToXls helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToXls()` with no return value.

[Visual Basic]

```
Public Sub ConfigureExportToXls()


End Sub
```

[end]

[C#]

```
private void ConfigureExportToXls()
{

}
```

[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection Excel.

[Visual Basic]

```
myExportOptions.ExportFormatType = ExportFormatType.Excel
```

[end]

[C#]

```
exportOptions.ExportFormatType = ExportFormatType.Excel;
```

[end]

5. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has an .xls file extension.

[Visual Basic]

```
   myDiskFileDestinationOptions.DiskFileName = exportPath & "Excel.xls"
```
[end]
[C#]
```
   diskFileDestinationOptions.DiskFileName = exportPath + "Excel.xls";
```
[end]

6. Finally, set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you have configured in the previous step.

[Visual Basic]
```
   myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```
[end]
[C#]
```
   exportOptions.DestinationOptions = diskFileDestinationOptions;
```
[end]

To create the ConfigureExportToPdf helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToPdf()` with no return value.

[Visual Basic]
```
   Public Sub ConfigureExportToPdf()


   End Sub
```
[end]
[C#]
```
   private void ConfigureExportToPdf()
   {
   }
```
[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection PortableDocFormat.

[Visual Basic]
```
   myExportOptions.ExportFormatType = ExportFormatType.PortableDocFormat
```
[end]
[C#]
```
   exportOptions.ExportFormatType = ExportFormatType.PortableDocFormat;
```
[end]

5. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has a .pdf file extension.

[Visual Basic]

```
myDiskFileDestinationOptions.DiskFileName = exportPath &
"PortableDoc.pdf"
```

[end]

[C#]

```
diskFileDestinationOptions.DiskFileName = exportPath + "PortableDoc.pdf";
```

[end]

6. Finally, set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you have configured in the previous step.

[Visual Basic]

```
myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```

[end]

[C#]

```
exportOptions.DestinationOptions = diskFileDestinationOptions;
```

[end]

To create the ConfigureExportToHtml32 helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToHtml32()` with no return value.

[Visual Basic]

```
Public Sub ConfigureExportToHtml32()


End Sub
```

[end]

[C#]

```
private void ConfigureExportToHtml32()
{
}
```

[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection HTML32.

[Visual Basic]

```
myExportOptions.ExportFormatType = ExportFormatType.HTML32
```

[end]

[C#]

```
exportOptions.ExportFormatType = ExportFormatType.HTML32;
```

[end]

5. Declare and instantiate the HTMLFormatOptions class with the variable name "html32FormatOptions."

[Visual Basic]

```
Dim html32FormatOptions As HTMLFormatOptions = New HTMLFormatOptions()
```

[end]

[C#]

```
HTMLFormatOptions html32FormatOptions = new HTMLFormatOptions();
```

[end]

6. Set the HTMLBaseFolderName property of the html32FormatOptions instance to the exportPath string, and the name "Html32Folder."

[Visual Basic]

```
html32FormatOptions.HTMLBaseFolderName = exportPath & "Html32Folder"
```

[end]

[C#]

```
html32FormatOptions.HTMLBaseFolderName = exportPath + "Html32Folder";
```

[end]

7. Set the HTMLFileName property of the html32FormatOptions instance to the name "html32.html."

[Visual Basic]

```
html32FormatOptions.HTMLFileName = "html32.html"
```

[end]

[C#]

```
html32FormatOptions.HTMLFileName = "html32.html";
```

[end]

8. Set the HTMLEnableSeparatedPage property of the html32FormatOptions instance to be "False."

[Visual Basic]

```
html32FormatOptions.HTMLEnableSeparatedPages = False
```

[end]

[C#]

```
html32FormatOptions.HTMLEnableSeparatedPages = false;
```

[end]

9. Set the HTMLHasPageNavigator property of the html32FormatOptions instance to be "False."

[Visual Basic]

```
html32FormatOptions.HTMLHasPageNavigator = False
```

[end]

[C#]

```
html32FormatOptions.HTMLHasPageNavigator = false;
```

[end]

10. Finally, assign the html32FormatOptions instance to the FormatOptions property of the ExportOptions instance.

[Visual Basic]

```
myExportOptions.FormatOptions = html32FormatOptions
```

[end]

[C#]

```
exportOptions.FormatOptions = html32FormatOptions;
```
[end]

To create the ConfigureExportToHtml40 helper method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method named `ConfigureExportToHtml40()` with no return value.

[Visual Basic]
```
Public Sub ConfigureExportToHtml40()


End Sub
```
[end]
[C#]
```
private void ConfigureExportToHtml40()
{
}
```
[end]

4. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection HTML40.

[Visual Basic]
```
myExportOptions.ExportFormatType = ExportFormatType.HTML40
```
[end]
[C#]
```
exportOptions.ExportFormatType = ExportFormatType.HTML40;
```
[end]

5. Declare and instantiate the HTMLFormatOptions class with the variable name "html40FormatOptions."

[Visual Basic]
```
Dim html40FormatOptions As HTMLFormatOptions = New HTMLFormatOptions()
```
[end]
[C#]
```
HTMLFormatOptions html40FormatOptions = new HTMLFormatOptions();
```
[end]

6. Set the HTMLBaseFolderName property of the html40FormatOptions instance to the exportPath string and the name "Html40Folder."

[Visual Basic]
```
html40FormatOptions.HTMLBaseFolderName = exportPath & "Html40Folder"
```
[end]
[C#]
```
html40FormatOptions.HTMLBaseFolderName = exportPath + "Html40Folder";
```
[end]

7. Set the HTMLFileName property of the html40FormatOptions instance to the name "html40.html."

[Visual Basic]

```
html40FormatOptions.HTMLFileName = "html40.html"
```

[end]

[C#]

```
html40FormatOptions.HTMLFileName = "html40.html";
```

[end]

8. Set the HTMLEnableSeparatedPage property of the html40FormatOptions instance to "True."

[Visual Basic]

```
html40FormatOptions.HTMLEnableSeparatedPages = True
```

[end]

[C#]

```
html40FormatOptions.HTMLEnableSeparatedPages = true;
```

[end]

9. Set the HTMLHasPageNavigator property of the html40FormatOptions instance to be "True."

[Visual Basic]

```
html40FormatOptions.HTMLHasPageNavigator = True
```

[end]

[C#]

```
html40FormatOptions.HTMLHasPageNavigator = true;
```

[end]

10. Set the FirstPageNumber property of the html40FormatOptions instance to 1.

[Visual Basic]

```
html40FormatOptions.FirstPageNumber = 1
```

[end]

[C#]

```
html40FormatOptions.FirstPageNumber = 1;
```

[end]

11. Set the LastPageNumber property of the html40FormatOptions instance 3.

[Visual Basic]

```
html40FormatOptions.LastPageNumber = 3
```

[end]

[C#]

```
html40FormatOptions.LastPageNumber = 3;
```

[end]

12. Finally, assign the html40FormatOptions instance to the FormatOptions property of the ExportOptions instance.

[Visual Basic]

```
    myExportOptions.FormatOptions = html40FormatOptions
```

[end]

[C#]

```
    exportOptions.FormatOptions = html40FormatOptions;
```

[end]

You have created the private helper methods that configure the multiple export formats.

Calling the Methods from the Case Statement

Earlier, you created a case statement in the ExportSelection() method, with multiple cases. Each case is linked to a selection from the ExportFormatType enum. You now call each of the configuration methods from the corresponding case.

To call the methods from the case statement

1. Locate the `ExportSelection()` method that you created earlier.
2. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.NoFormat, set the selectedNoFormat Boolean variable to True.

[Visual Basic]

```
    selectedNoFormat = true
```

[end]

[C#]

```
    selectedNoFormat = true;
```

[end]

3. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.CrystalReport, call the `ConfigureExportToRpt()` method.

[Visual Basic]

```
    ConfigureExportToRpt()
```

[end]

[C#]

```
    ConfigureExportToRpt();
```

[end]

4. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.RichText, call the `ConfigureExportToRtf()` method.

[Visual Basic]

```
    ConfigureExportToRtf()
```

[end]

[C#]

```
    ConfigureExportToRtf();
```

[end]

5. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.WordForWindows, call the `ConfigureExportToDoc()` method.

[Visual Basic]

```
    ConfigureExportToDoc()
```

[end]

[C#]

```
ConfigureExportToDoc();
```

[end]

6. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.Excel, call the `ConfigureExportToXls()` method.

[Visual Basic]

```
ConfigureExportToXls()
```

[end]

[C#]

```
ConfigureExportToXls();
```

[end]

7. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.PortableDocFormat, call the `ConfigureExportToPdf()` method.

[Visual Basic]

```
ConfigureExportToPdf()
```

[end]

[C#]

```
ConfigureExportToPdf();
```

[end]

8. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.HTML32, call the `ConfigureExportToHtml32()` method.

[Visual Basic]

```
ConfigureExportToHtml32()
```

[end]

[C#]

```
ConfigureExportToHtml32();
```

[end]

9. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.HTML40, call the `ConfigureExportToHtml40()` method.

[Visual Basic]

```
ConfigureExportToHtml40()
```

[end]

[C#]

```
ConfigureExportToHtml40();
```

[end]

If you are creating a project in Visual Studio 2005 or Crystal Reports Developer you must complete the procedures in Creating Methods for the New Exporting Formats, before you continue to Calling the Methods to Perform the Export.

Calling the Methods to Perform the Export

You are now ready to create the button click event method for the exportByType Button control, and then call the methods to perform the export from that event method.

To create the exportByType_Click event method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. Double-click the exportByType Button control.

   The `exportByType_Click()` event method is created and displayed in Code view.
4. Within the `exportByType_Click()` event method, enter calls to the three event methods that you created earlier to perform the export.

[Visual Basic]

```
ExportSetup()

ExportSelection()

ExportCompletion()
```

[end]

[C#]

```
ExportSetup();

ExportSelection();

ExportCompletion();
```

[end]

You are now ready to build and run your project, and then export your report in multiple formats.

To test the project

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The Web or Windows application compiles and displays the Hierarchical Grouping report. A **DropDownList** control shows multiple export formats, and an **Export As Selected Type** button is displayed to perform the export.
4. Select one of the export formats from the **DropDownList** control, and then click the **Export As Selected Type** button.

   A message appears next to the button that indicates whether the export has been successful. If not successful, an error message is shown.
5. Try to export in several different formats.
6. Return to Visual Studio and click **Stop** to exit from debug mode.
7. Check the Exported file directory on your Web server or Windows machine to confirm that the exported files have been placed in the directory.

Conclusion

You have successfully created an application that can programmatically export your report in multiple export formats.

To learn about exporting with enhanced API features, continue to Addendum: Enhancements to the Exporting Code.

Addendum: Enhancements to the Exporting Code

If you have installed Visual Studio 2005 or Crystal Reports Developer you have access to the enhanced API for exporting your Crystal report to multiple formats.

In the enhanced Crystal Reports API, the ExportFormatTypes enum has two new exporting formats: Microsoft Excel records (an .xls file that contains only the data) and text files. To learn how to add code for the new exporting formats, see Creating Methods for the New Exporting Formats.

In addition, you now have the following enhanced exporting methods in the ReportDocument class:

ExportToDisk(CrystalDecisions.Shared.ExportFormatType formatType, string fileName): exports a report to the local drive of the Web server or to the Windows machine.

For more information, see Using the ExportToDisk() Method.

ExportToHttpResponse(CrystalDecisions.Shared.ExportFormatType formatType, System.Web.HttpResponse response, bool asAttachment, string attachmentName): exports a report in the specified format type to a browser window, or it exports the report as an attachment.

For more information, see Using the ExportToHttpResponse() Method.

ExportToHttpResponse(CrystalDecisions.Shared.ExportOptions options, System.Web.HttpResponse response, bool asAttachment, string attachmentName): exports a report in the specified format type to a browser window, or it exports the report as an attachment.

For more information, see Using the ExportToHttpResponse() Method.

ExportToStream(CrystalDecisions.Shared.ExportFormatType formatType): exports a report's data in the specified format type to an input/output stream.

For more information, see Using the ExportToStream() Method.

Creating Methods for the New Exporting Formats

In this section, you learn how to modify your project (that you have created earlier in this tutorial), to include the new exporting formats.

Before you modify the code to include the new exporting formats, complete all the procedures in Exporting to Multiple Formats.

To modify the code to include the new exporting formats

1. Open the completed project for this tutorial.
2. Open the Web or Windows Form.
3. From the **View** menu, click **Code**.

4. Within the "Select Case" [Visual Basic] or "switch" [C#] statement of the ExportSelection() method, add a case statement for the ExcelRecord and the Text formats.

[Visual Basic]

```
Case ExportFormatType.ExcelRecord


Case ExportFormatType.Text
```

[end]

[C#]

```
case ExportFormatType.ExcelRecord:
   break;
case ExportFormatType.Text:
   break;
```

[end]

Create the private helper methods that configure the multiple export formats. First, you create the ConfigureExportToXlsRec() method to set the ExcelRecord export options.

5. At the bottom of the class, create a private helper method named ConfigureExportToXlsRec() with no return value.

[Visual Basic]

```
Public Sub ConfigureExportToXlsRec()


End Sub
```

[end]

[C#]

```
private void ConfigureExportToXlsRec()
{
}
```

[end]

6. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection ExcelRecord.

[Visual Basic]

```
myExportOptions.ExportFormatType = ExportFormatType.ExcelRecord
```

[end]

[C#]

```
exportOptions.ExportFormatType = ExportFormatType.ExcelRecord;
```

[end]

7. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has an .xls file extension.

[Visual Basic]

```
myDiskFileDestinationOptions.DiskFileName = exportPath &
"ExcelRecord.xls"
```

[end]

[C#]

```
diskFileDestinationOptions.DiskFileName = exportPath + "ExcelRecord.xls";
```

[end]

8. Set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you configured in the previous step.

[Visual Basic]

```
myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```

[end]

[C#]

```
exportOptions.DestinationOptions = diskFileDestinationOptions;
```

[end]

9. At the bottom of the class, create a private helper method named `ConfigureExportToTxt()` with no return value. This method is used to set the text export options.

[Visual Basic]

```
Public Sub ConfigureExportToTxt()


End Sub
```

[end]

[C#]

```
private void ConfigureExportToTxt()
{
}
```

[end]

10. Within the method, set the ExportFormatType property of the ExportOptions instance to the ExportFormatType enum selection Text.

[Visual Basic]

```
myExportOptions.ExportFormatType = ExportFormatType.Text
```

[end]

[C#]

```
exportOptions.ExportFormatType = ExportFormatType.Text;
```

[end]

11. Set the DiskFileName property of the DiskFileDestinationOptions instance to the exportPath string, and then follow it with the name of a document that has a .txt file extension.

[Visual Basic]

```
myDiskFileDestinationOptions.DiskFileName = exportPath & "Text.txt"
```

[end]

[C#]

```
diskFileDestinationOptions.DiskFileName = exportPath + "Text.txt";
```

[end]

12. Set the DestinationOptions property of the ExportOptions instance to the DiskFileDestinationOptions instance that you configured in the previous step.

[Visual Basic]

```
myExportOptions.DestinationOptions = myDiskFileDestinationOptions
```

[end]

[C#]

```
exportOptions.DestinationOptions = diskFileDestinationOptions;
```

[end]

You now call the export configuration methods from the corresponding case.

13. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.ExcelRecord, call the `ConfigureExportToXlsRec()` method.

[Visual Basic]

```
ConfigureExportToXlsRec()
```

[end]

[C#]

```
ConfigureExportToXlsRec();
```

[end]

14. Within the `Select Case [Visual Basic]` or `switch [C#]` case statement, in the case for ExportFormatType.Text, call the `ConfigureExportToTxt()` method.

[Visual Basic]

```
ConfigureExportToTxt()
```

[end]

[C#]

```
ConfigureExportToTxt();
```

[end]

You have successfully added two new exporting formats to your project.

To further explore the enhanced API, choose one of the following enhanced exporting methods:

If you want to minimize your code and do not need to set ExportOptions, continue to Using the ExportToDisk() Method.

For a Web Site, if you want to export your report to a browser window, or export the report as an attachment in a browser window, continue to Using the ExportToHttpResponse() Method.

If you want pass your report's data to an input/output stream, continue to Using the ExportToStream() Method.

Using the ExportToDisk() Method

The ExportToDisk() method takes an ExportFormatType and a file name parameter. This method simplifies the need to use the export configuration methods that you created in Creating Methods That Configure Multiple Export Formats.

In this section, you learn how to create a new project, how to modify an existing project, and how to add code to the project to use the ExportToDisk() method.

Prerequisites:

You must create a project that is based on the instructions in Setting Up a Project for the ExportToDisk() Method.

Or, you must create a project that is based on the instructions in Creating Methods for the New Exporting Formats.

Then, you need to modify the project as shown in Preparing the Project for the ExportToDisk() Method.

Once you have prepared the project, click the appropriate link to jump to that section:

Modifying the ExportSetup() Method

Calling the ExportToDisk() Method

Setting Up a Project for the ExportToDisk() Method

In this section, you learn how to create a new project for the ExportToDisk() method.

To set up a new project for the ExportToDisk() Method

1. Complete the instructions in Adding Controls to the Web or Windows Form.
2. Create the `ExportSetup()` method and the `ExportSelection()` method in Creating Three Methods That Perform the Export.
3. Within the `"Select Case" [Visual Basic]` or `"switch" [C#]` statement of the `ExportSelection()` method, add a case statement for the ExcelRecord and the Text formats.

[Visual Basic]

```
Case ExportFormatType.ExcelRecord


Case ExportFormatType.Text
```

[end]

[C#]

```
case ExportFormatType.ExcelRecord:
   break;
case ExportFormatType.Text:
   break;
```

[end]

4. Create a conditional block to test the Boolean variable, selectedNoFormat.

[Visual Basic]

```
If selectedNoFormat Then
```

```
    Else


    End If
```
[end]
[C#]
```
    if (selectedNoFormat)
    {
    }
    else
    {
    }
```
[end]

5. Within the If block, set the Text property of the message Label control to the FORMAT_NOT_SUPPORTED constant of the MessageConstants class.

   Note You created the MessageConstants class for this tutorial in Additional Setup Requirements of Appendix: Project Setup.

[Visual Basic]
```
    message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
```
[end]
[C#]
```
    message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
```
[end]

6. Within the Else block, set the Text property of the message Label control to the SUCCESS constant of the MessageConstants class.

[Visual Basic]
```
    message.Text = MessageConstants.SUCCESS
```
[end]
[C#]
```
    message.Text = MessageConstants.SUCCESS;
```
[end]

7. Create a try/catch block with the Exception class that is referenced as a variable named "ex." The try block encloses the `"Select Case" [Visual Basic]` or `"switch" [C#]` statement and the conditional block.

[Visual Basic]
```
    Try


    Catch ex As Exception
```

```
    End Try
```
[end]

[C#]

```
    try
    {
    }
    catch (Exception ex)
    {
    }
```
[end]

8. Within the catch block, set the Text property of the message Label control to the FAILURE constant of the MessagesConstants class, and then append to it the Message property of the Exception parameter.

[Visual Basic]

```
    message.Text = MessageConstants.FAILURE & ex.Message
```
[end]

[C#]

```
    message.Text = MessageConstants.FAILURE + ex.Message;
```
[end]

9. Outside the try/catch block, set the Visible property of the message Label control to "True."

[Visual Basic]

```
    message.Visible = True
```
[end]

[C#]

```
    message.Visible = true;
```
[end]

10. From the **View** menu, click **Designer**.
11. Double-click the exportByType Button control.

    The `exportByType_Click()` event method is created, and you are taken to the Code view.

12. Within the `exportByType_Click()` event method, enter calls to the ExportSetup() and ExportSelection() methods.

[Visual Basic]

```
    ExportSetup()
    ExportSelection()
```
[end]

[C#]

```
    ExportSetup();
    ExportSelection();
```

[end]

Preparing the Project for the ExportToDisk() Method

In this section, you learn how to modify a project that is a result of the procedure in Creating Methods for the New Exporting Formats.

Now, you must delete certain lines of code that are not needed for the ExportToDisk() method.

To modify the project to use the ExportToDisk() method

1. Open the project.
2. Open the Web or Windows Form.
3. From the **View** menu, click **Code**.
4. At the top of the class, delete the following class declarations:

[Visual Basic]

```
Private myDiskFileDestinationOptions As DiskFileDestinationOptions
Private myExportOptions As ExportOptions
```

[end]

[C#]

```
private DiskFileDestinationOptions diskFileDestinationOptions;
private ExportOptions exportOptions;
```

[end]

5. Within the `ExportSetup()` method, delete all the lines of code after the conditional block. (The last line of code that calls the FormatOptions property only occurs in a Windows project.)

[Visual Basic]

```
myDiskFileDestinationOptions = New DiskFileDestinationOptions()
myExportOptions = hierarchicalGroupingReport.ExportOptions
myExportOptions.ExportDestinationType = ExportDestinationType.DiskFile
myExportOptions.FormatOptions = Nothing
```

[end]

[C#]

```
diskFileDestinationOptions = new DiskFileDestinationOptions();
exportOptions = hierarchicalGroupingReport.ExportOptions;
exportOptions.ExportDestinationType = ExportDestinationType.DiskFile;
exportOptions.FormatOptions = null;
```

[end]

6. Delete the following export configuration methods:

```
ConfigureExportToRpt()
ConfigureExportToRtf()
ConfigureExportToDoc()
ConfigureExportToXls()
```

```
ConfigureExportToPdf()
ConfigureExportToHtml32()
ConfigureExportToHtml40()
ConfigureExportToXlsRec()
ConfigureExportToTxt()
```

7. Within the `Select Case [Visual Basic]` or `switch [C#]` case statements of `ExportSelection()`, delete the calls to the export configuration methods.
8. Within the `ExportCompletion()` method, delete the call to the `Export()` method.
9. Copy and paste all the code from the `ExportCompletion()` method to the top of the `ExportSelection()` method.
10. Delete the `ExportCompletion()` method and delete the call to `ExportCompletion()` in the exportByType button click event.
11. Within the `ExportSelection()` method, cut and paste the `Select Case [Visual Basic]` or `switch [C#]` case statements above the If block within the try block.

    The try/catch block now looks like the following:

[Visual Basic]

```
Try
    Select Case exportTypesList.SelectedIndex
        Case ExportFormatType.NoFormat
            selectedNoFormat = True
        Case ExportFormatType.CrystalReport

        Case ExportFormatType.RichText

        Case ExportFormatType.WordForWindows

        Case ExportFormatType.Excel

        Case ExportFormatType.PortableDocFormat

        Case ExportFormatType.HTML32

        Case ExportFormatType.HTML40

    End Select
    If selectedNoFormat Then
        message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
```

```
        Else
            message.Text = MessageConstants.SUCCESS
        End If
    Catch ex As Exception
        message.Text = MessageConstants.FAILURE & ex.Message
    End Try
[end]
[C#]
    try
    {
       switch ((ExportFormatType)exportTypesList.SelectedIndex)
       {
             case ExportFormatType.NoFormat:
                   selectedNoFormat = true;
                   break;
             case ExportFormatType.CrystalReport:
                   break;
             case ExportFormatType.RichText:
                   break;
             case ExportFormatType.WordForWindows:
                   break;
             case ExportFormatType.Excel:
                   break;
             case ExportFormatType.PortableDocFormat:
                   break;
             case ExportFormatType.HTML32:
                   break;
             case ExportFormatType.HTML40:
                   break;
             case ExportFormatType.ExcelRecord:
                   break;
             case ExportFormatType.Text:
                   break;
       }
       if (selectedNoFormat)
```

```
        {
              message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
        }
        else
        {
              message.Text = MessageConstants.SUCCESS;
        }
    }
    catch (Exception ex)
    {
        message.Text = MessageConstants.FAILURE + ex.Message;
    }
```

[end]

Modifying the ExportSetup() Method

In this section, you learn how to modify the ExportSetup() method to create folders for the HTML32 and HTML40 export formats.

To modify the ExportSetup() method

1. At the top of the class, add the following class declarations:

[Visual Basic]

```
    Private exportPathHTML32 As String
    Private exportPathHTML40 As String
```

[end]

[C#]

```
    private string exportPathHTML32;
    private string exportPathHTML40;
```

[end]

2. Within the `ExportSetup()` method, instantiate the exportPathHTML32 and exportPathHTML40 string variables to the root directory of the hard drive.

[Visual Basic]

```
    exportPathHTML32 = "C:\Exported\HTML32\"
    exportPathHTML40 = "C:\Exported\HTML40\"
```

[end]

[C#]

```
    exportPathHTML32 = "C:\\Exported\\HTML32\\";
    exportPathHTML40 = "C:\\Exported\\HTML40\\";
```

[end]

Note If you want to place the folders within the Web directory of your Web server, prefix the folder name with the Request.PhysicalApplicationPath property.

3. Create a conditional block that tests whether the directory in the exportPathHTML32 string already exists.

[Visual Basic]

```
If Not System.IO.Directory.Exists(exportPathHTML32) Then


End If
```

[end]

[C#]

```
if (!System.IO.Directory.Exists(exportPathHTML32))
{
}
```

[end]

4. Within the conditional block, call the `CreateDirectory()` method of System.IO.Directory to create the directory in the exportPathHTML32 string.

[Visual Basic]

```
System.IO.Directory.CreateDirectory(exportPathHTML32)
```

[end]

[C#]

```
System.IO.Directory.CreateDirectory(exportPathHTML32);
```

[end]

5. Create a conditional block that tests whether the directory in the exportPathHTML40 string already exists.

[Visual Basic]

```
If Not System.IO.Directory.Exists(exportPathHTML40) Then


End If
```

[end]

[C#]

```
if (!System.IO.Directory.Exists(exportPathHTML40))
{
}
```

[end]

6. Within the conditional block, call the `CreateDirectory()` method of System.IO.Directory to create the directory in the exportPathHTML40 string.

[Visual Basic]

```
System.IO.Directory.CreateDirectory(exportPathHTML40)
```

[end]

[C#]

```
System.IO.Directory.CreateDirectory(exportPathHTML40);
```

[end]

Calling the ExportToDisk() Method

In this section, you learn to call the ExportToDisk() method in each case statement of the ExportSelection() method.

To call the ExportToDisk() method in the ExportSelection() method

1. Within the `ExportSelection()` method, declare a string variable and instantiate the variable to an empty string.

[Visual Basic]

```
Dim myFileName As String = ""
```

[end]

[C#]

```
string fileName = "";
```

[end]

2. Within the ExportFormatType.CrystalReport case statement, do the following:
   a) Set the file name string to the exportPath string, and then follow it with the name of a document that has an .rpt file extension.

[Visual Basic]

```
myFileName = exportPath & "Report.rpt"
```

[end]

[C#]

```
fileName = exportPath + "Report.rpt";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.CrystalReport and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.CrystalRepo
rt, myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.CrystalRepo
rt, fileName);
```

[end]

3. Within the ExportFormatType.RichText case statement, do the following:
   a) Set the file name string to the exportPath string, and then follow it with the name of a document that uses an .rtf file extension.

[Visual Basic]

```
myFileName = exportPath & "RichTextFormat.rtf"
```

[end]

[C#]

```
fileName = exportPath + "RichTextFormat.rtf";
```

[end]

b. Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.RichText and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.RichText,
myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.RichText,
fileName);
```

[end]

4. Within the ExportFormatType.WordForWindows case statement, do the following:
   a) Set the file name string to the exportPath string, and then follow it with the name of a document that uses a .doc file extension.

[Visual Basic]

```
myFileName = exportPath & "Word.doc"
```

[end]

[C#]

```
fileName = exportPath + "Word.doc";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.WordForWindows and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.WordForWind
ows, myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.WordForWind
ows, fileName);
```

[end]

5. Within the ExportFormatType.Excel case statement, do the following:
   a) Set the file name string to the exportPath string, and then follow it with the name of a document that uses an .xls file extension.

[Visual Basic]

```
myFileName = exportPath & "Excel.xls"
```

[end]

[C#]

```
fileName = exportPath + "Excel.xls";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.Excel and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.Excel,
myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.Excel,
fileName);
```

[end]

6. Within the ExportFormatType.PortableDocFormat case statement, do the following:
   a) Set the file name string to the exportPath string, and then follow it with the name of a document that uses a .pdf file extension.

[Visual Basic]

```
myFileName = exportPath & "PortableDoc.pdf"
```

[end]

[C#]

```
fileName = exportPath + "PortableDoc.pdf";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.PortableDocFormat and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.PortableDoc
Format, myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.PortableDoc
Format, fileName);
```

[end]

7. Within the ExportFormatType.HTML32 case statement, do the following:
   a) Set the file name string to the exportPathHTML32 string, and then follow it with the name of a document that uses an .html file extension.

[Visual Basic]

```
myFileName = exportPathHTML32 & "HTML32.html"
```

[end]

[C#]

```
fileName = exportPathHTML32 + "HTML32.html";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.HTML32 and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.HTML32,
myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.HTML32,
fileName);
```

[end]

8. Within the ExportFormatType.HTML40 case statement, do the following:
   a) Set the file name string to the exportPathHTML40 string, and then follow it with the name of a document that uses an .html file extension.

[Visual Basic]

```
myFileName = exportPathHTML40 & "HTML40.html"
```

[end]

[C#]

```
fileName = exportPathHTML40 + "HTML40.html";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.HTML40 and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.HTML40,
myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.HTML40,
fileName);
```

[end]

9. Within the ExportFormatType.ExcelRecord case statement, do the following:
   a) Set the file name string to the exportPath string, and then follow it with the name of a document that uses an .xls file extension.

[Visual Basic]

```
myFileName = exportPath & "ExcelRecord.xls"
```

[end]

[C#]

```
fileName = exportPath + "ExcelRecord.xls";
```

[end]

   b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.ExcelRecord and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.ExcelRecord
, myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.ExcelRecord
, fileName);
```

[end]

10. Within the ExportFormatType.Text case statement, do the following:
    a) Set the file name string to the exportPath string, and then follow it with the name of a document that uses a .txt file extension.

[Visual Basic]

```
myFileName = exportPath & "Text.txt"
```

[end]

[C#]

```
fileName = exportPath + "Text.txt";
```

[end]

    b) Call the `ExportToDisk()` method from the hierarchicalGroupingReport instance and pass in ExportFormatType.Text and the file name string.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.Text,
myFileName)
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToDisk(ExportFormatType.Text,
fileName);
```

[end]

You are now ready to build and run the project, to export your Crystal report into different formats.

If you want use the other enhanced API methods, click the appropriate link to jump to that section:

Using the ExportToHttpResponse() Method

Using the ExportToStream() Method

Using the ExportToHttpResponse() Method

The ExportToHttpResponse() method allows you to export your Crystal report to a browser window, or to export the report as an attachment. This method is used only for Web Sites.

The available overloaded methods for ExportToHttpResponse() include the following:

ExportToHttpResponse(CrystalDecisions.Shared.ExportFormatType formatType, System.Web.HttpResponse response, bool asAttachment, string attachmentName)

ExportToHttpResponse(CrystalDecisions.Shared.ExportOptions options, System.Web.HttpResponse response, bool asAttachment, string attachmentName)

The ExportToHttpResponse() method does not support exports to HTTP Response for the HTML32 and HTML40 formats. Therefore, when you try to export to HTML32 or HTML40, an error message appears.

If the asAttachment Boolean variable is set to True, a File Download dialog box appears. If the asAttachment Boolean variable is set to False, the exported report opens in the browser window.

When you choose to save the file, the file name is set to the attachmentName string variable. If you do not specify the attachmentName variable, then the default file name is

"Untitled," with the specified file extension. The file name can be changed in the Save As dialog box.

Click the appropriate link to jump to that section:

Using the ExportToHttpResponse() Method with the ExportFormatType Enumeration

Using the ExportToHttpResponse() Method with the ExportOptions Class

Using the ExportToHttpResponse() Method with the ExportFormatType Enumeration

To use the ExportToHttpResponse() method with the ExportFormatType parameter, you must complete the procedure Adding Controls to the Web or Windows Form for a Web Site. You do not need the ExportSetup(), ExportSelection(), and ExportCompletion() method. The required code is placed in the exportByType click event method.

To use the ExportToHttpResponse() method with the ExportFormatType parameter

1. Open the Web Form.
2. From the **View** menu, click **Designer**.
3. Double-click the exportByType Button control.

   The `exportByType_Click()` event method is created, and you are taken to the Code view.
4. At the top of the class, add an ExportOptions class declaration.

[Visual Basic]

```
Private myExportOptions As ExportOptions
```

[end]

[C#]

```
private ExportOptions exportOptions;
```

[end]

5. Within the `exportByType_Click()` event method, instantiate the ExportOptions instance.

[Visual Basic]

```
myExportOptions = New ExportOptions()
```

[end]

[C#]

```
exportOptions = new ExportOptions();
```

[end]

6. Create a try/catch block with the Exception class that is referenced as a variable named "ex."

[Visual Basic]

```
Try


Catch ex As Exception
```

```
    End Try
[end]
[C#]
    try
    {
    }
    catch (Exception ex)
    {
    }
[end]
```

7. Within the try block, create a conditional block to test if the selected item from the exportTypesList is equal to ExportFormatType.NoFormat.

```
[Visual Basic]
    If (exportTypesList.SelectedIndex = ExportFormatType.NoFormat) Then


    Else


    End If
[end]
[C#]
    if ((ExportFormatType)exportTypesList.SelectedIndex ==
    ExportFormatType.NoFormat)
    {
    }
    else
    {
    }
[end]
```

8. Within the If block, set the Text property of the message Label control to the FORMAT_NOT_SUPPORTED constant of the MessageConstants class.

    Note You created the MessageConstants class for this tutorial in Additional Setup Requirements of Appendix: Project Setup.

```
[Visual Basic]
    message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
[end]
[C#]
    message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
[end]
```

9. Within the Else block, assign the selected ExportFormatType from the exportTypesList to the ExportFormatType property of the ExportOptions instance.

[Visual Basic]

```
myExportOptions.ExportFormatType = exportTypesList.SelectedIndex
```

[end]

[C#]

```
exportOptions.ExportFormatType =
(ExportFormatType)exportTypesList.SelectedIndex;
```

[end]

10. Within the Else block, call the `ExportToHttpResponse()` method of the hierarchicalGroupingReport instance. Pass in the ExportOptions instance, the ASP.NET Response object, a True Boolean value, and a file name string as the method parameters..

    Note The file name string for the attachment does not need an extension because the extension is automatically added to the exported file.

[Visual Basic]

```
hierarchicalGroupingReport.ExportToHttpResponse(myExportOptions,
Response, True, "ExportedReport")
```

[end]

[C#]

```
hierarchicalGroupingReport.ExportToHttpResponse(exportOptions,
Response, True, "ExportedReport");
```

[end]

11. Within the catch block, set the Text property of the message Label control to the FAILURE constant of the MessagesConstants class, and then append to it the Message property of the Exception parameter.

[Visual Basic]

```
message.Text = MessageConstants.FAILURE & ex.Message
```

[end]

[C#]

```
message.Text = MessageConstants.FAILURE + ex.Message;
```

[end]

12. Outside the try/catch block, set the Visible property of the message Label control to "True."

[Visual Basic]

```
message.Visible = True
```

[end]

[C#]

```
message.Visible = true;
```

[end]

You are now ready to build and run the project, to export your Crystal report into different formats.

If you want to use the ExportToHttpResponse() method with the ExportOptions class, see Using the ExportToHttpResponse() Method with the ExportOptions Class.

To use the other enhanced API methods, click the appropriate link to jump to that section:

Using the ExportToDisk() Method

Using the ExportToStream() Method

Using the ExportToHttpResponse() Method with the ExportOptions Class

To use the ExportToHttpResponse() method with the ExportOptions parameter, you must complete procedure Adding Controls to the Web or Windows Form for a Web Site. You do not need the ExportSetup(), ExportSelection(), and ExportCompletion() method. The required code is placed in the exportByType click event method.

To use the ExportToHttpResponse() method with the ExportOptions parameter

1. Open the Web Form.
2. From the **View** menu, click **Designer**.
3. Double-click the exportByType Button control.

   The `exportByType_Click()` event method is created, and you are taken to the Code view.
4. Within the `exportByType_Click()` event method, create a try/catch block with the Exception class that is referenced as a variable named "ex."

[Visual Basic]

```
Try


Catch ex As Exception


End Try
```

[end]

[C#]

```
try
{
}
catch (Exception ex)
{
}
```

5. Within the try block, create a conditional block to test if the selected item from the exportTypesList is equal to ExportFormatType.NoFormat.

[Visual Basic]

```
If (exportTypesList.SelectedIndex = ExportFormatType.NoFormat) Then
```

```
    Else


    End If
```
[end]
[C#]
```
    if ((ExportFormatType)exportTypesList.SelectedIndex ==
    ExportFormatType.NoFormat)
    {
    }
    else
    {
    }
```
[end]

6. Within the If block, set the Text property of the message Label control to the FORMAT_NOT_SUPPORTED constant of the MessageConstants class.

    Note The If condition is satisfied when the NoFormat value from the DropDownList is selected.

[Visual Basic]
```
    message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
```
[end]
[C#]
```
    message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
```
[end]

7. Within the Else block, call the `ExportToHttpResponse()` method of the hierarchicalGroupingReport instance. Pass in the selected ExportFormatType, the ASP.NET Response object, a True Boolean value, and a file name string as the method parameters.

    Note The file name string for the attachment does not need an extension because the extension is automatically added to the exported file.

[Visual Basic]
```
    hierarchicalGroupingReport.ExportToHttpResponse(exportTypesList.Selecte
    dIndex, Response, True, "ExportedReport")
```
[end]
[C#]
```
    hierarchicalGroupingReport.ExportToHttpResponse((ExportFormatType)expor
    tTypesList.SelectedIndex, Response, True, "ExportedReport");
```
[end]

8. Within the catch block, set the Text property of the message Label control to the FAILURE constant of the MessagesConstants class, and then append to it the Message property of the Exception parameter.

[Visual Basic]

```
    message.Text = MessageConstants.FAILURE & ex.Message
```

[end]

[C#]

```
    message.Text = MessageConstants.FAILURE + ex.Message;
```

[end]

9. Outside the try/catch block, set the Visible property of the message Label control to "True."

[Visual Basic]

```
    message.Visible = True
```

[end]

[C#]

```
    message.Visible = true;
```

[end]

You are now ready to build and run the project, to export your Crystal report into different formats.

To use the other enhanced API methods, click the appropriate link to jump to that section:

Using the ExportToDisk() Method

Using the ExportToStream() Method

Using the ExportToStream() Method

In this section, you learn how to use the ExportToStream() method to export the report to the input/output stream as a sequence of bytes. Then, you learn how to write the sequence of bytes to a file of your specified format.

When you export the report to the HTML formats, the images are not exported. It is recommended to use the ExportToHttpResponse() methods when you want to export to the HTML formats.

Prerequisites:

You must create a project that is based on the instructions in Setting Up a Project for the ExportToStream() Method.

Or, you must create a project that is based on the instructions in Creating Methods for the New Exporting Formats.

Then, you need to modify the project as shown in Preparing the Project for the ExportToStream() Method.

Once you have prepared the project, click the appropriate link to jump to that section:

Modifying the Case Statements in the ExportSelection() Method

Calling the ExportToStream() Method

Setting Up a Project for the ExportToStream() Method

In this section, you learn how to create a new project for the ExportToStream() method.

To set up a new project for the ExportToStream() Method

1. Complete the instructions in Adding Controls to the Web or Windows Form.
2. Create the `ExportSetup()` method and the `ExportSelection()` method in Creating Three Methods That Perform the Export.
3. Within the `"Select Case" [Visual Basic]` or `"switch" [C#]` statement of the `ExportSelection()` method, add a case statement for the ExcelRecord and the Text formats.

[Visual Basic]

```
Case ExportFormatType.ExcelRecord


Case ExportFormatType.Text
```

[end]

[C#]

```
case ExportFormatType.ExcelRecord:
   break;
case ExportFormatType.Text:
   break;
```

[end]

4. Create a conditional block to test the Boolean variable, selectedNoFormat.

[Visual Basic]

```
If selectedNoFormat Then


Else


End If
```

[end]

[C#]

```
if (selectedNoFormat)
{
}
else
{
}
```

[end]

5. Within the If block, set the Text property of the message Label control to the FORMAT_NOT_SUPPORTED constant of the MessageConstants class.

   Note You have created the MessageConstants class for this tutorial in Additional Setup Requirements of Appendix: Project Setup.

[Visual Basic]

```
   message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
```
[end]
[C#]
```
   message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
```
[end]

6. Within the Else block, set the Text property of the message Label control to the SUCCESS constant of the MessageConstants class.

[Visual Basic]
```
   message.Text = MessageConstants.SUCCESS
```
[end]
[C#]
```
   message.Text = MessageConstants.SUCCESS;
```
[end]

7. Create a try/catch block with the Exception class that is referenced as a variable named "ex." The try block encloses the `"Select Case" [Visual Basic]` or `"switch" [C#]` statement and the conditional block.

[Visual Basic]
```
   Try


   Catch ex As Exception


   End Try
```
[end]
[C#]
```
   try
   {
   }
   catch (Exception ex)
   {
   }
```
[end]

8. Within the catch block, set the Text property of the message Label control to the FAILURE constant of the MessagesConstants class, and then append to it the Message property of the Exception parameter.

[Visual Basic]
```
   message.Text = MessageConstants.FAILURE & ex.Message
```
[end]
[C#]
```
   message.Text = MessageConstants.FAILURE + ex.Message;
```
[end]

9. Outside the try/catch block, set the Visible property of the message Label control to "True."

[Visual Basic]

```
message.Visible = True
```

[end]

[C#]

```
message.Visible = true;
```

[end]

10. From the **View** menu, click **Designer**.
11. Double-click the exportByType Button control.

    The `exportByType_Click()` event method is created, and you are taken to the Code view.
12. Within the `exportByType_Click()` event method, enter calls to the ExportSetup() and ExportSelection() methods.

[Visual Basic]

```
ExportSetup()
ExportSelection()
```

[end]

[C#]

```
ExportSetup();
ExportSelection();
```

[end]

Preparing the Project for the ExportToStream() Method

In this section, you learn how to modify a project that is a result of the procedure in Creating Methods for the New Exporting Formats.

Now, you must delete certain lines of code that are not needed for the ExportToStream() method.

To modify the project to use the ExportToStream() method

1. Open the project.
2. Open the Web or Windows Form.
3. From the **View** menu, click **Code**.
4. At the top of the class, delete the following class declarations:

[Visual Basic]

```
Private myDiskFileDestinationOptions As DiskFileDestinationOptions
Private myExportOptions As ExportOptions
```

[end]

[C#]

```
private DiskFileDestinationOptions diskFileDestinationOptions;
```

```
private ExportOptions exportOptions;
```

[end]

5. Within the `ExportSetup()` method, delete all the lines of code after the conditional block. (The last line of code that calls the FormatOptions property only occurs in a Windows project.)

[Visual Basic]

```
myDiskFileDestinationOptions = New DiskFileDestinationOptions()
myExportOptions = hierarchicalGroupingReport.ExportOptions
myExportOptions.ExportDestinationType = ExportDestinationType.DiskFile
myExportOptions.FormatOptions = Nothing
```

[end]

[C#]

```
diskFileDestinationOptions = new DiskFileDestinationOptions();
exportOptions = hierarchicalGroupingReport.ExportOptions;
exportOptions.ExportDestinationType = ExportDestinationType.DiskFile;
exportOptions.FormatOptions = null;
```

[end]

6. Delete the following export configuration methods:

```
ConfigureExportToRpt()
ConfigureExportToRtf()
ConfigureExportToDoc()
ConfigureExportToXls()
ConfigureExportToPdf()
ConfigureExportToHtml32()
ConfigureExportToHtml40()
ConfigureExportToXlsRec()
ConfigureExportToTxt()
```

7. Within the `Select Case [Visual Basic]` or `switch [C#]` case statements of `ExportSelection()`, delete the calls to the export configuration methods.
8. Within the `ExportCompletion()` method, delete the call to the Export() method.
9. Copy and paste all the code from the `ExportCompletion()` method to the top of the `ExportSelection()` method.
10. Delete the `ExportCompletion()` method and delete the call to `ExportCompletion()` in **exportByType** button click event.
11. Within the `ExportSelection()` method, cut and paste the `Select Case [Visual Basic]` or `switch [C#]` case statements above the If block within the try block.

    The try/catch block now looks like the following:

[Visual Basic]

```
Try
    Select Case exportTypesList.SelectedIndex
```

```
            Case ExportFormatType.NoFormat
                selectedNoFormat = True
            Case ExportFormatType.CrystalReport

            Case ExportFormatType.RichText

            Case ExportFormatType.WordForWindows

            Case ExportFormatType.Excel

            Case ExportFormatType.PortableDocFormat

            Case ExportFormatType.HTML32

            Case ExportFormatType.HTML40

        End Select
        If selectedNoFormat Then
            message.Text = MessageConstants.FORMAT_NOT_SUPPORTED
        Else
            message.Text = MessageConstants.SUCCESS
        End If
    Catch ex As Exception
        message.Text = MessageConstants.FAILURE & ex.Message
    End Try
[end]
[C#]
    try
    {
       switch ((ExportFormatType)exportTypesList.SelectedIndex)
       {
            case ExportFormatType.NoFormat:
                  selectedNoFormat = true;
                  break;
            case ExportFormatType.CrystalReport:
```

```
                  break;
            case ExportFormatType.RichText:
                  break;
            case ExportFormatType.WordForWindows:
                  break;
            case ExportFormatType.Excel:
                  break;
            case ExportFormatType.PortableDocFormat:
                  break;
            case ExportFormatType.HTML32:
                  break;
            case ExportFormatType.HTML40:
                  break;
            case ExportFormatType.ExcelRecord:
                  break;
            case ExportFormatType.Text:
                  break;
         }
         if (selectedNoFormat)
         {
               message.Text = MessageConstants.FORMAT_NOT_SUPPORTED;
         }
         else
         {
               message.Text = MessageConstants.SUCCESS;
         }
      }
      catch (Exception ex)
      {
         message.Text = MessageConstants.FAILURE + ex.Message;
      }
[end]
```

Modifying the Case Statements in the ExportSelection() Method

In this section, you learn how to set a file name string for each ExportFormatType case statement.

To modify the case statements in the ExportSelection() method

1. Within the `ExportSelection()` method, declare a string variable and instantiate the variable to an empty string.

[Visual Basic]

```
Dim myFileName As String = ""
```

[end]

[C#]

```
string fileName = "";
```

[end]

2. Within the ExportFormatType.CrystalReport case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .rpt file extension.

[Visual Basic]

```
myFileName = exportPath & "Report.rpt"
```

[end]

[C#]

```
myFileName = exportPath + "Report.rpt";
```

[end]

3. Within the ExportFormatType.RichText case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .rtf file extension.

[Visual Basic]

```
myFileName = exportPath & "RichTextFormat.rtf"
```

[end]

[C#]

```
myFileName = exportPath + "RichTextFormat.rtf";
```

[end]

4. Within the ExportFormatType.WordForWindows case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .doc file extension.

[Visual Basic]

```
      myFileName = exportPath & "Word.doc"
```

[end]

[C#]

```
      fileName = exportPath + "Word.doc";
```

[end]

5. Within the ExportFormatType.Excel case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .xls file extension.

[Visual Basic]

```
myFileName = exportPath & "Excel.xls"
```

[end]

[C#]

```
fileName = exportPath + "Excel.xls";
```

[end]

6. Within the ExportFormatType.PortableDocFormat case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .pdf file extension.

[Visual Basic]

```
myFileName = exportPath & "PortableDoc.pdf"
```

[end]

[C#]

```
fileName = exportPath + "PortableDoc.pdf";
```

[end]

7. Within the ExportFormatType.HTML32 case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .html file extension.

[Visual Basic]

```
myFileName = exportPath & "HTML32.html"
```

[end]

[C#]

```
fileName = exportPath + "HTML32.html";
```

[end]

8. Within the ExportFormatType.HTML40 case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .html file extension.

[Visual Basic]

```
myFileName = exportPath & "HTML40.html"
```

[end]

[C#]

```
fileName = exportPath + "HTML40.html";
```

[end]

9. Within the ExportFormatType.ExcelRecord case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .xls file extension.

[Visual Basic]

```
myFileName = exportPath & "ExcelRecord.xls"
```

[end]

[C#]

```
fileName = exportPath + "ExcelRecord.xls";
```

[end]

10. Within the ExportFormatType.Text case statement, set the file name string to the exportPath string that is followed by a recognizable document name with a .txt file extension.

[Visual Basic]

```
myFileName = exportPath & "Text.txt"
```

[end]

[C#]

```
fileName = exportPath + "Text.txt";
```

[end]

Calling the ExportToStream() Method

In this section, you learn how to call the ExportToStream() method and to write the exported report data to a file in your specified format.

To call the ExportToStream() method in the ExportSelection() method

1. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class for the System.IO namespace.

[Visual Basic]

```
Imports System.IO
```

[end]

[C#]

```
using System.IO;
```

[end]

2. Within the Else block of the `ExportSelection()` method, call the `ExportToStream()` method of the hierarchicalGroupingReport instance, pass in the selected ExportFormatType from the exportTypesList dropdownlist, and assign the instance to the Stream class.

[Visual Basic]

```
Stream myStream =
hierarchicalGroupingReport.ExportToHttpResponse(exportTypesList.Selecte
dIndex)
```

[end]

[C#]

```
Stream stream =
hierarchicalGroupingReport.ExportToHttpResponse((ExportFormatType)expor
tTypesList.SelectedIndex);
```

[end]

3. Create a new byte array that has the same length as the Stream instance.

[Visual Basic]

```
Dim myDataArray As byte() = New byte(myStream.Length)
```

[end]

[C#]

```
byte[] dataArray = new byte[stream.Length];
```

[end]

4. Read the data from the Stream instance to the byte array from a zero offset to the length of the Stream instance.

[Visual Basic]

```
myStream.Read(myDataArray, 0, Convert.ToInt32(myStream.Length));
```

[end]

[C#]

```
stream.Read(dataArray, 0, Convert.ToInt32(stream.Length));
```

[end]

5. Create a FileStream instance that creates the file specified by the file name string variable.

[Visual Basic]

```
Dim myFileStream As FileStream = New FileStream(myFileName,
System.IO.FileMode.Create)
```

[end]

[C#]

```
FileStream fileStream = new FileStream(fileName,
System.IO.FileMode.Create);
```

[end]

6. Write the data stored in the byte array to the file from a zero offset to the length of the byte array.

[Visual Basic]

```
myFileStream.Write(myDataArray, 0, myDataArray.Length)
```

[end]

[C#]

```
fileStream.Write(dataArray, 0, dataArray.Length);
```

[end]

7. Close the FileStream instance and the Stream instance.

[Visual Basic]

```
myFileStream.Close()
myStream.Close()
```

[end]

[C#]

```
fileStream.Close();
stream.Close();
```

[end]

8. Set the Text property of the message Label control to the SUCCESS constant of the MessageConstants class.

[Visual Basic]

```
message.Text = MessageConstants.SUCCESS
```

[end]

[C#]

```
message.Text = MessageConstants.SUCCESS;
```

[end]

You are now ready to build and run the project, to export your Crystal report into different formats.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_Export

C# Windows project: CS_Win_RDObjMod_Export

Visual Basic Web Site: VB_Web_RDObjMod_Export

Visual Basic Windows project: VB_Win_RDObjMod_Export

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Printing and Setting Print Options

Printing and Setting Print Options

Introduction

In this tutorial, you learn how to configure print options and send a print command from code.

You can configure print options and call a printer from code, rather than from the Crystal Reports UI. To do that, you use the PrintOptions class and the PrintToPrinter() method of ReportDocument object model.

If the Print button on the toolbar of the CrystalReportViewer control meets your printing needs, then you do not need to write code to configure additional print options.

However, a code-based approach to printing reports is useful in specialized scenarios:

You can control when, where, and how printing occurs. To do that, you disable the Print button in the toolbar of the CrystalReportViewer control and manage all printing through code.

You can print a report in the background, without displaying it. All of the Print settings are contained within the ReportDocument model, which can be instantiated and configured without ever displaying the report with a CrystalReportViewer control.

You can centralize all printing on the Web server for a Web client. Use the PrintToPrinter() method to send print jobs to a printer that is connected to the Web server, rather than send print jobs to a local printer that is connected to the Web client.

Note This back-end printing functionality is less extensive than the report-scheduling framework that is provided with Crystal Reprots Server or BusinessObjects Enterprise.

To begin this tutorial, you add several list controls to set print options above the CrystalReportViewer control on your Web or Windows Form.

Then, in the code-behind class, you create a CURRENT_PRINTER string constant, to which you assign the path of the printer that you want to use.

Next, you bind each list control to an enumeration that contains print options for paper orientation, paper size, and printer duplex settings. An additional list control displays custom paper source settings for the printer that is currently designated in the CURRENT_PRINTER constant. For the custom paper source control, you create a helper method that instantiates PrinterSettings, assigns it to the current printer, and then returns an array of paper sources for that printer.

Then you create a button click event method for the Print Report button. In that event method, each print option property is assigned a value that is based on selections made in the list controls. Finally, the report is printed to the printer that is designated in the CURRENT_PRINTER constant.

Adding Print Option Controls

In this section, you add controls to display various print options above the CrystalReportViewer control on the Web or Windows Form.

Choose from one (but not both) of the step procedures below.

To add controls to the Web Form

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix:Project Setup.

1. Right click on the web form in the **Solution Explorer** and click **View Designer**.
2. Click the **CrystalReportViewer** control to select it.
3. Press the LEFT ARROW key so that a flashing cursor appears, and then press ENTER five times.

   The **CrystalReportViewer** control drops by five lines.
4. On the first line, type "Paper Orientation."
5. From the **Toolbox**, drag a **DropDownList** control to the right of the text on the first line.

   Note If a Smart Task panel appears on the DropDownList controls (if you use Visual Studio 2005), press Esc to close it.
6. On the second line, type "Paper Size."
7. From the **Toolbox**, drag a **DropDownList** control to the right of the text on the second line.
8. On the third line, type "Printer Duplex."
9. From the **Toolbox**, drag a **DropDownList** control to the right of the text on the third line.
10. On the fourth line, type "Paper Source."
11. From the **Toolbox**, drag a **DropDownList** control to the right of the text on the fourth line.
12. From the **Toolbox**, drag a **Button** control onto the fifth line.
13. From the **Toolbox**, drag a **Label** control to the right of the **Button** control on the fifth line.

To add controls to the Windows Form

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. Open the Windows Form in Design View.
2. Click the **CrystalReportViewer** control to select it.
3. From the **Properties** window, set **Dock** to "Bottom."
4. Resize the Windows Form so that you leave enough room above the **CrystalReportViewer** control for additional controls.
5. From the **Toolbox**, drag a **Label** control to the top left of the Windows Form.
6. From the **Properties** window, set the **Text** property of the **Label** control to "Paper Orientation."
7. From the **Toolbox**, drag a **ComboBox** control to the right of the **Label** control.

8. From the **Toolbox**, drag a second **Label** control directly beneath the first **Label** control.
9. From the **Properties** window, set the **Text** property of the second **Label** control to "Paper Size."
10. From the **Toolbox**, drag a **ComboBox** control to the right of the second **Label** control.
11. From the **Toolbox**, drag a third **Label** control directly beneath the second **Label** control.
12. From the **Properties** window, set the **Text** property of the third **Label** control to "Printer Duplex."
13. From the **Toolbox**, drag a **ComboBox** control to the right of the third **Label** control.
14. From the **Toolbox**, drag a fourth **Label** control directly beneath the third **Label** control.
15. From the **Properties** window, set the **Text** property of the fifth **Label** control to " Paper Source."
16. From the **Toolbox**, drag a **ComboBox** control to the right of the fourth **Label** control.
17. From the **Toolbox**, drag a **Button** control directly beneath the fourth **Label** control.
18. From the **Toolbox**, drag a fifth **Label** control to the right of the **Button** control.

Setting Properties for the Print Option Controls

In this section, you set properties for the Print Option controls on the Web or Windows Form.

To set properties for the Print Option controls

1. On the first line, click the DropDownList (ComboBox) control to select it.
2. From the **Properties** window, set the **ID (Name)** property to "paperOrientationList."
3. On the second line, click the **DropDownList (ComboBox)** control to select it.
4. From the **Properties** window, set the **ID (Name)** property to "paperSizeList."
5. On the third line, click the **DropDownList (ComboBox)** control to select it.
6. From the **Properties** window, set the **ID (Name)** property to "printerDuplexList."
7. On the fourth line, click the **DropDownList (ComboBox)** control to select it.
8. From the **Properties** window, set the **ID (Name)** property to "paperSourceList."
9. On the fifth line, click the **Button** control to select it.
10. From the **Properties** window, do the following:

    Set the **Name(ID)** property to "printReport."

    Set the **Text** property to "Print Report From Server" (for a Web Site) or "Print Report" (for a Windows project).

    Note You may need to resize the Button control.
11. On the fifth line, click the **Label** control to select it.
12. From the **Properties** window, do the following:

    Set the **Name(ID)** property to "message."

    Set the **Text** property to be blank.

13. From the **File** menu, click **Save All**.

Populating the Print Option Controls

In this section, you populate the DropDownList (or ComboBox) controls on the Web or Windows Form. To do that, you write code in the code-behind class.

Most controls are easily populated from the values of printer enumerations in the CrystalDecisions.Shared namespace. However, one of these controls, paperSourceList, must be populated with custom paper sources that are based on the currently selected printer. So, your first step is to designate a current printer, and then create a helper method that generates an ArrayList of custom paper sources from that printer.

To create the GetPaperSources() helper method

1. Open the Web Form.
2. From the **View** menu, click **Code**.
3. At the top of the class, create a string constant for the network path of the printer.

   Note In this example, the printer path "\\NetworkPrintServer2\\Printer15" is used.

[Visual Basic]

```
Private Const CURRENT_PRINTER As String =
"\\NetworkPrintServer2\Printer15"
```

[end]

[C#]

```
private const string CURRENT_PRINTER =
@"\\NetworPrinterServer2\Printer15";
```

[end]

4. Above the class, add an `"Imports"[Visual Basic]` or `"using"[C#]` statement for the System.Collections namespace.

[Visual Basic]

```
Imports System.Collections
```

[end]

[C#]

```
using System.Collections;
```

[end]

5. At the bottom of the class, create the `GetPaperSources()` helper method that returns an ArrayList.

[Visual Basic]

```
Private Function GetPaperSources() As ArrayList

End Function
```

[end]

[C#]

```
private ArrayList GetPaperSources()
{
```

```
    }
```
[end]

The rest of the code in this step procedure goes into the GetPaperSources() method.

6. Within the method, declare and instantiate an ArrayList.

[Visual Basic]
```
    Dim myArrayList As ArrayList = New ArrayList()
```
[end]
[C#]
```
    ArrayList arrayList = new ArrayList();
```
[end]

7. Declare and instantiate the PrinterSettings class from the System.Drawing.Printing namespace.

    Note Several of the classes that are used in this tutorial have recurring names, in both the System.Drawing.Printing namespace and the CrystalDecisions.Shared namespace. When a class that has a recurring name is used in the tutorial, the full namespace precedes the class name to remove ambiguity about which namespace it belongs to.

[Visual Basic]
```
    Dim myPrinterSettings As System.Drawing.Printing.PrinterSettings = New
    System.Drawing.Printing.PrinterSettings()
```
[end]
[C#]
```
    System.Drawing.Printing.PrinterSettings printerSettings = new
    System.Drawing.Printing.PrinterSettings();
```
[end]

8. Set the PrinterName property of the PrinterSettings instance to the CURRENT_PRINTER string constant.

[Visual Basic]
```
    myPrinterSettings.PrinterName = CURRENT_PRINTER
```
[end]
[C#]
```
    printerSettings.PrinterName = CURRENT_PRINTER;
```
[end]

9. Create a foreach loop that loops through each PaperSource instance in the PaperSources indexed class instance.

[Visual Basic]
```
    Dim myPaperSource As System.Drawing.Printing.PaperSource
    For Each myPaperSource As System.Drawing.Printing.PaperSource In
    myPrinterSettings.PaperSources
    Next
```
[end]
[C#]

```
foreach (System.Drawing.Printing.PaperSource paperSource in
printerSettings.PaperSources)
{
}
```

[end]

10. Within the foreach loop, add the SourceName property of the PaperSource instance to the ArrayList.

[Visual Basic]

```
myArrayList.Add(myPaperSource.SourceName.ToString())
```

[end]

[C#]

```
arrayList.Add(paperSource.SourceName.ToString());
```

[end]

11. Outside the foreach loop, return the ArrayList from the method.

[Visual Basic]

```
Return myArrayList
```

[end]

[C#]

```
return arrayList;
```

[end]

12. From the **File** menu, click **Save All**.

Now, you must populate the first three DropDownList controls from enumerations in the CrystalDecisions.Shared namespace. The fourth DropDownList control is populated from the GetPaperSources() method that you have just created.

To populate the DropDownList controls

1. Open the Web or Windows Form.
2. From the View menu, click Designer.
3. Double-click on any empty area on the form.

   The page switches to Code view and generates a Page_Load() event method (for a Web Form) or a Form1_Load() event method (for a Windows Form).
4. If you are developing a Web Site, within the `Page_Load()` event method, create a `Not IsPostBack` conditional block.

[Visual Basic]

```
If Not IsPostBack Then
End If
```

[end]

[C#]

```
if (!IsPostBack)
{
}
```

[end]

Note The `Not IsPostBack` conditional block encapsulates code that is only run the first time the page loads. Controls are typically bound to data values within `Not IsPostBack` conditional blocks, so that their data values (and any subsequent control events) are not reset during page reloads.

5. The following lines of code are placed differently for Windows projects compared to Web Sites:

   In Windows projects, the following lines of code should be placed in the Form_Load event handler.

   In Web Sites, the following lines of code should be nested within the Not IsPostBack conditional block within the Page_Load event handler.

   Set the DataSource property of the paperOrientationList control instance to the values of the PaperOrientation enum.

[Visual Basic]

```
paperOrientationList.DataSource =
System.Enum.GetValues(GetType(PaperOrientation))
```

[end]

[C#]

```
paperOrientationList.DataSource =
System.Enum.GetValues(typeof(PaperOrientation));
```

[end]

6. Set the DataSource property of the paperSizeList control instance to the values of the PaperSize enum.

[Visual Basic]

```
paperSizeList.DataSource = System.Enum.GetValues(GetType(PaperSize))
```

[end]

[C#]

```
paperSizeList.DataSource = System.Enum.GetValues(typeof(PaperSize));
```

[end]

7. Set the DataSource property of the printerDuplexList control instance to the values of the PrinterDuplex enum.

[Visual Basic]

```
printerDuplexList.DataSource =
System.Enum.GetValues(GetType(PrinterDuplex))
```

[end]

[C#]

```
printerDuplexList.DataSource =
System.Enum.GetValues(typeof(PrinterDuplex));
```

[end]

8. Set the DataSource property of the paperSourceList control instance to `GetPaperSources()` helper method that you created in the previous section.

[Visual Basic]

```
        paperSourceList.DataSource = GetPaperSources()
```

[end]

[C#]

```
        paperSourceList.DataSource = GetPaperSources();
```

[end]

9. Finally, if you are building a Web Site, enter a call to the `DataBind()` method, to bind the values of the four DropDownList controls.

[Visual Basic]

```
        DataBind()
```

[end]

[C#]

```
        DataBind();
```

[end]

Retrieving the Selected Paper Source

The paperSourceList control that you added to the Web or Windows Form displays a list of custom paper sources, based on the currently selected printer. When the end user selects a paper source from the paperSourceList control at runtime, this selected paper source must be applied to the CustomPaperSource property of the report.

But, only two possible value types can be retrieved from the paperSourceList control:

The String value for the selected item.

The Integer index of the selected item.

Neither type (String or Integer) is compatible with the CustomPaperSource property. It can only be assigned the type System.Drawing.Printing.PaperSource.

So, in this section you create a helper method, GetSelectedPaperSource() that determines and then returns the correct PaperSource instance, based on the selected index of the paperSourceList control.

To do that, the method loops through the PaperSources collection for the currently selected printer, and then compares the SourceName string property of the PaperSource instance against the string value for the selected item. When the matching PaperSource is found, that PaperSource instance is returned from the method.

To create the GetSelectedPaperSource() method

1. At the bottom of the class, create the `GetSelectedPaperSource()` helper method that returns a PaperSource instance.

[Visual Basic]

```
    Private Function GetSelectedPaperSource() As
    System.Drawing.Printing.PaperSource

    End Function
```

[end]

[C#]

```
    private System.Drawing.Printing.PaperSource GetSelectedPaperSource()

    {
```

```
    }
```
[end]

The rest of the code in this step procedure goes into the GetSelectedPaperSource() method.

2. Within the method, declare and instantiate the PaperSource class from the System.Drawing.Printing namespace.

[Visual Basic]
```
Dim selectedPaperSource As System.Drawing.Printing.PaperSource = New
System.Drawing.Printing.PaperSource
```
[end]
[C#]
```
System.Drawing.Printing.PaperSource selectedPaperSource = new
System.Drawing.Printing.PaperSource();
```
[end]

3. Declare and instantiate the PrinterSettings class from the System.Drawing.Printing namespace.

[Visual Basic]
```
Dim myPrinterSettings As System.Drawing.Printing.PrinterSettings = New
System.Drawing.Printing.PrinterSettings()
```
[end]
[C#]
```
System.Drawing.Printing.PrinterSettings printerSettings = new
System.Drawing.Printing.PrinterSettings();
```
[end]

4. Assign to the PrinterName property of the PrinterSettings instance the string constant CURRENT_PRINTER.

[Visual Basic]
```
myPrinterSettings.PrinterName = CURRENT_PRINTER
```
[end]
[C#]
```
printerSettings.PrinterName = CURRENT_PRINTER;
```
[end]

5. Create a foreach loop that loops through each PaperSource instance in the PaperSources indexed class instance.

[Visual Basic]
```
For Each myPaperSource As System.Drawing.Printing.PaperSource In
myPrinterSettings.PaperSources

Next
```
[end]
[C#]

```
foreach (System.Drawing.Printing.PaperSource paperSource in
printerSettings.PaperSources)
{
}
```

[end]

6. Within the foreach loop, add a conditional block that tests whether the SourceName property of the PaperSource instance matches the selected item in the paperSourceList control.

   Note The name of that selected item is different for the DropDownList control in a Web Site compared to a ComboBox control in a Windows project. Complete the appropriate instructions that are listed below.

   a) In a Windows project, create the conditional block, which specifies the name of the selected item in the paperSourceList control as the SelectedText property.

[Visual Basic]

```
If myPaperSource.SourceName = paperSourceList.SelectedText Then
End If
```

[end]

[C#]

```
if (paperSource.SourceName == paperSourceList.SelectedText)
{
}
```

[end]

   b) Or, in a Web Site, create the conditional block, which specifies the name of the selected item in the paperSourceList control as the SelectedItem.Text property.

[Visual Basic]

```
If myPaperSource.SourceName = paperSourceList.SelectedItem.Text Then
End If
```

[end]

[C#]

```
if (paperSource.SourceName == paperSourceList.SelectedItem.Text)
{
}
```

[end]

7. Within the conditional block, assign the PaperSource instance of the current foreach loop iteration to the selectedPaperSource instance that was declared at the top of the method.

[Visual Basic]

```
selectedPaperSource = myPaperSource
```

[end]

[C#]

```
selectedPaperSource = paperSource;
```

[end]

8. Outside the conditional block, and outside the foreach loop, return the selectedPaperSource instance from the method.

[Visual Basic]

```
Return selectedPaperSource
```

[end]

[C#]

```
return selectedPaperSource;
```

[end]

Setting the Print Options

In this section, you learn how to create the SetPrintOptions() helper method. In this method, you populate multiple properties of the PrintOptions instance. Some of those properties are populated directly from control selections: one is assigned from the CURRENT_PRINTER string constant, and one is assigned the return value of the GetSelectedPaperSource() method, which you created in the previous section.

To create the SetPrintOptions() method

1. At the bottom of the class, create the `SetPrintOptions()` helper method.

[Visual Basic]

```
Private Sub SetPrintOptions()

End Sub
```

[end]

[C#]

```
private void SetPrintOptions()
{
}
```

[end]

   The rest of the code in this step procedure is entered into the SetPrintOptions() method.

2. Within the method, declare and instantiate PrintOptions, and then assign it to the PrintOptions property of the report instance.

[Visual Basic]

```
Dim myPrintOptions As PrintOptions =
hierarchicalGroupingReport.PrintOptions
```

[end]

[C#]

```
PrintOptions printOptions = hierarchicalGroupingReport.PrintOptions;
```

[end]

3. Set the PrinterName property of the PrintOptions instance to the CURRENT_PRINTER string constant.

[Visual Basic]

```
myPrintOptions.PrinterName = CURRENT_PRINTER
```

[end]

[C#]

```
printOptions.PrinterName = CURRENT_PRINTER;
```

[end]

4. Set the PaperOrientation property of the PrintOptions instance to the PaperOrientation enum selection that is retrieved from the paperOrientationList control.

[Visual Basic]

```
myPrintOptions.PaperOrientation =
CType(paperOrientationList.SelectedIndex, PaperOrientation)
```

[end]

[C#]

```
printOptions.PaperOrientation =
(PaperOrientation)paperOrientationList.SelectedIndex;
```

[end]

5. Set the PaperSize property of the PrintOptions instance to the PaperSize enum selection that is retrieved from the paperSizeList control.

[Visual Basic]

```
myPrintOptions.PaperSize = CType(paperSizeList.SelectedIndex, PaperSize)
```

[end]

[C#]

```
printOptions.PaperSize = (PaperSize)paperSizeList.SelectedIndex;
```

[end]

6. Set the PrinterDuplex property of the PrintOptions instance to the PrinterDuplex enum selection that is retrieved from the printerDuplexList control.

[Visual Basic]

```
myPrintOptions.PrinterDuplex = CType(printerDuplexList.SelectedIndex,
PrinterDuplex)
```

[end]

[C#]

```
printOptions.PrinterDuplex =
(PrinterDuplex)printerDuplexList.SelectedIndex;
```

[end]

7. Set the CustomPaperSource property of the PrintOptions instance to the `GetSelectedPaperSource()` helper method that you created earlier.

[Visual Basic]

```
myPrintOptions.CustomPaperSource = GetSelectedPaperSource()
```

[end]

[C#]

```
printOptions.CustomPaperSource = GetSelectedPaperSource();
```

[end]

Configuring the Print Button Click Event Method

In this section, you learn how to create the Print button click event method, configure print options, and call the print job within this event.

To configure the print button click event method

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. Double-click the **printReport** Button control.

   The code-behind class is displayed, where a `redisplay_Click()` event method has been automatically generated.
4. Within the `printReport_Click()` event method, call the **`SetPrintOptions()`** helper method that you created in the previous section.

[Visual Basic]

```
SetPrintOptions()
```

[end]

[C#]

```
SetPrintOptions();
```

[end]

5. Create a try/catch block.

[Visual Basic]

```
Try

Catch ex As Exception

End Try
```

[end]

[C#]

```
try
{
}
catch (Exception ex)
{
}
```

[end]

6. Within the try block, call the `PrintToPrinter()` method to print one non-collated copy, from page 1 to 99.

[Visual Basic]

```
hierarchicalGroupingReport.PrintToPrinter(1, False, 1, 99)
```

[end]

[C#]

```
hierarchicalGroupingReport.PrintToPrinter(1, false, 1, 99);
```

[end]

7. Still within the try block, set the Text property of the message Label instance to the MessageConstants.SUCCESS string constant.

   Note You created the MessageConstants class during Project Setup. If you did not, you must create this class before proceeding. See Add a Class for Error Messages.

[Visual Basic]

```
message.Text = MessageConstants.SUCCESS
```

[end]

[C#]

```
message.Text = MessageConstants.SUCCESS;
```

[end]

8. Within the catch block, set the Text property of the message Label instance to the MessageConstants.FAILURE string constant. Append to it the Message property of the Exception instance.

[Visual Basic]

```
message.Text = MessageConstants.FAILURE & ex.Message
```

[end]

[C#]

```
message.Text = MessageConstants.FAILURE + ex.Message;
```

[end]

You are now ready to build and run your project.

To test the printing of the report

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The report displays along with a selection of print options.
4. Make a selection in each of the print option controls, and then click the **Print Report** button.

   The message Label control will provide either a success or failure message. If the message is a failure, check the CURRENT_PRINTER constant and other settings for errors.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

In this tutorial, you learned how to configure print options.

You placed controls on the form and populated them to display standard print options from printing enumerations in the CrystalDecisions.Shared class, as well as custom paper sources for the currently selected printer.

You then wrote code in the button click event, to set each print option and to send the print job to the printer.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_SetPrintOptions

C# Windows project: CS_Win_RDObjMod_SetPrintOptions

Visual Basic Web Site: VB_Web_RDObjMod_SetPrintOptions

Visual Basic Windows project: VB_Win_RDObjMod_SetPrintOptions

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Filtering Data Using Selection Formulas

Filtering Data Using Selection Formulas

Introduction

In this tutorial, you learn how to filter data within the ReportDocument object model by setting the RecordSelectionFormula property of the DataDefinition class.

Selection formulas are used to filter records that you want to display on a Crystal report. To write selection formulas, you can use the Basic syntax and the Crystal syntax.

In this tutorial, you create a selection formula to filter customer records where the Last Year's Sales field is greater than a specific value, and the Customer Name field is compared to another string. A DropDownList (Web) or ComboBox (Windows) control selects a comparison operator for the Customer Name field. You can choose to display customer names that are equal to, less than, greater than, less than or equal to, greater than or equal to, or not equal to the string value that you have specified.

The formula is passed as a string variable to the SelectionFormula property of the CrystalReportViewer class. Once the property is set, the Crystal report that binds to the CrystalReportViewer control is filtered before it is displayed.

This tutorial can also be completed with classes of the CrystalReportViewer object model, although the ReportDocument object model is preferred.

Creating a Report

To begin, you create a Crystal report where you will filter data.

To create a Crystal report

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the template **Crystal Report**.
3. In the **Name** field, enter the name "CustomerBySalesName.rpt" and click **Add**.
4. If you have not registered before, you are asked to register. To find out how to register, see Appendix: Crystal Reports Registration and Keycode.
5. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
6. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
7. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.
8. From the subfolder that opens, expand the **ODBC (RDO)** folder.
9. In the **ODBC (RDO)** window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

The ODBC (RDO) folder expands and shows the Xtreme Sample Database.

10. Expand the **Tables** node and select the **Customer** table.
11. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
12. CTRL-click **Customer Name**, **Last Year's Sales**, and **City**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Finish** button.

    The CustomerBySalesName report is created and loaded into the main window of Visual Studio.
14. Click **Main Report Preview**.

    The report is previewed with all available data. Later you filter this data programmatically.
15. Click **Main Report** to exit preview mode.

Binding the Report

In Appendix: Project Setup, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous step, you have added a CustomerBySalesName report to the project.

In this section, you instantiate the CustomerBySalesName report and bind it to the CrystalReportViewer control.

You can instantiate and bind the report in two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the report as a non-embedded report.

To instantiate the CustomerBySalesName report as an embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the CustomerBySalesName report wrapper class, using the variable name customerBySalesNameReport. Set its access modifier to private.

[Visual Basic]

```
Private customerBySalesNameReport As CustomerBySalesName
```

[end]

[C#]

```
private CustomerBySalesName customerBySalesNameReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
customerBySalesNameReport = New CustomerBySalesName()
```

[end]

[C#]

```
customerBySalesNameReport = new CustomerBySalesName();
```

[end]

5. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer control to the instantiated report class (variable name: customerBySalesNameReport).

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customerBySalesNameReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customerBySalesNameReport;
```

[end]

You are now ready to build and run your project. Skip to the next section below.

To instantiate the CustomerBySalesName report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, with the variable name customerBySalesNameReport. Set its access modifier to private.

[Visual Basic]

```
Private customerBySalesNameReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument customerBySalesNameReport;
```

[end]

   Note The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
customerBySalesNameReport = New ReportDocument()
```

[end]

[C#]

```
customerBySalesNameReport = new ReportDocument();
```

[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("CustomerBySalesName.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("CustomerBySalesName.rpt");
```

[end]

   For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

   Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"CustomerBySalesName.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" +
"CustomerBySalesName.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
customerBySalesNameReport.Load(reportPath)
```

[end]

[C#]

```
customerBySalesNameReport.Load(reportPath);
```

[end]

7. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customerBySalesNameReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customerBySalesNameReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: customerBySalesNameReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project.

To test the loading of the CustomerBySalesName report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The report is displayed, and it shows all available rows of data. In the next section, you start to filter that data with the RecordSelectionFormula property.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Applying a Filter to the Report Data

In this section, you learn how to apply a filter to the data in the report.

To apply a filter to report data

1. Open the Web or Windows form in Design view.
2. From the **View** menu, click **Code**.
3. At the top of the class, add three new class-level variable declarations that will represent values by which data will be filtered.

[Visual Basic]

```
Private salesAmount As String
Private operatorValue As String
Private customerName As String
```

[end]

[C#]

```
private string salesAmount;
private string operatorValue;
private string customerName;
```

[end]

4. If you are creating a Windows project, add an additional class-level variable declaration, a boolean that is named useDefaultValues, and then assign to it a value of True.

[Visual Basic]

```
Private useDefaultValues As Boolean = True
```

[end]

[C#]

```
private bool useDefaultValues = true;
```

[end]

5. Within the `ConfigureCrystalReports()` method, create a conditional block.

   If you are creating a Web Site, create a Not IsPostBack conditional block. Place this block above the existing code in the method.

[Visual Basic]

```
If Not IsPostBack Then

End If
```

[end]

[C#]

```
if (!IsPostBack)
{

}
```

[end]

   If you are creating a Windows project, create a useDefaultValues conditional block. Place this block above the existing code in the method.

[Visual Basic]

```
If useDefaultValues Then

End If
```

[end]

[C#]

```
if (useDefaultValues)
{

}
```

[end]

6. Within the conditional block, assign values of 11000, =, and A to the three class-level variables.

   These values are assigned within a conditional block, because later you override these values with a button click event.

[Visual Basic]

```
salesAmount = "11000"
operatorValue = "="
customerName = "A"
```

[end]

[C#]

```
salesAmount = "11000";
operatorValue = "=";
customerName = "A";
```

[end]

7. Immediately outside and below the conditional block, create a selectionFormula string in which you mix literals with the class-level variables you have just created.

[Visual Basic]

```
Dim selectionFormula As String = "{Customer.Last Year's Sales} > " _
    & salesAmount _
    & " AND Mid({Customer.Customer Name}, 1, 1) " _
    & operatorValue _
    & "'" _
    & customerName _
    & "'"
```

[end]

[C#]

```
string selectionFormula = "{Customer.Last Year's Sales} > "
    + salesAmount
    + " AND Mid({Customer.Customer Name}, 1, 1) "
    + operatorValue
    + "'"
    + customerName
    + "'";
```

[end]

8. In the report binding code that you created earlier, create a line break just above the line of code where customerBySalesName is assigned to the ReportSource property of the CrystalReportViewer control.
9. In this line break, assign the selectionFormula string variable to the ReportDocument instance.

[Visual Basic]

```
customerBySalesNameReport.DataDefinition.RecordSelectionFormula =
selectionFormula
```

[end]

[C#]

```
customerBySalesNameReport.DataDefinition.RecordSelectionFormula =
selectionFormula;
```

[end]

10. Compile and view your application.

    The data should now be filtered.

11. Return to Visual Studio and click **Stop** to exit from debug mode.

In the next section, you create controls on the form that allow you to adjust the selection formula dynamically.

Adding Controls for Dynamic Filtering

In this section, you add controls to use in the selection formula. These controls allow you to filter the data dynamically.

To add controls to use in the selection formula

1. Open the Web or Windows Form.
2. From the **View** menu, click **Designer**.
3. If you are developing a Web Site, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   b) Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER three times.

The CrystalReportViewer control drops by three lines.

4. If you are developing a Windows project, do the following:
   a) Click the **CrystalReportViewer** control to select it.
   c) From the **Properties** window, set **Dock** to "Bottom".
   d) Resize the **CrystalReportViewer** control, so that approximately three lines appear above it.
   e) From the **Properties** window, set **Anchor** to "Top, Bottom, Left, Right".
5. From the **Toolbox**, drag a **Label** control onto the form at top left.
6. In the **Properties** window, set the **Text** to "Display the following customers:"
7. From the **Toolbox**, drag a second **Label** control onto the form below the first one.
8. In the **Properties** window, set the **Text** to "- last year's sales > $"
9. From the **Toolbox**, drag a third **Label** control onto the form below the second one.
10. In the **Properties** window, set the **Text** to "- first letter of name is"
11. From the **Toolbox**, drag a **TextBox** control to the right of the second **Label** control.
12. In the **Properties** window, set the **ID** to "lastYearsSales".
13. From the **Toolbox**, drag a **DropDownList** control(Web) or **ComboBox** control(Windows) to the right of the third **Label** control.
14. In the **Properties** window, set the **ID** to "operatorValueList".
15. From the **Toolbox**, drag a **TextBox** control to the right of the **DropDownList** control(Web) or **ComboBox** control(Windows).
16. In the **Properties** window, set the **ID** to "letterOfName".
17. From the **Toolbox**, drag a **Button** control below the third **Label** control.
18. In the **Properties** window, set the **ID** to "redisplay" and the **Text** to "Redisplay Report".
19. From the **Toolbox**, drag a **Label** control below the **Button** control.

20. In the **Properties** window, set the **ID** to "formula" and the **Text** to be blank.

    Ensure that the CrystalReportViewer control is adjusted to sit immediately below these controls.

21. From the **File** menu, click **Save All**.

In the next section, you populate the DropDownList (Web) or ComboBox (Windows) control.

Populating the DropDownList ComboBox Control

The DropDownList control (Web) or ComboBox control (Windows) requires a list of operatorValues (equal, greater than, and so on). The most generic way to populate this control is to provide the list of values as an enumerator.

To populate the DropDownList ComboBox control

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, select **Class**.
3. Enter the name "CeComparisonOperator" and then click OK.
4. In the class file, change the word class to enum.

   Note In Visual Basic, remember to change both the opening and closing signature. In C#, delete the constructor.

5. Enter the following enum values.

[Visual Basic]

```
        EqualTo
        GreaterThan
        GreaterThanOrEqualTo
        LessThan
        LessThanOrEqualTo
        NotEqualTo
```

[end]

[C#]

```
        EqualTo,
        GreaterThan,
        GreaterThanOrEqualTo,
        LessThan,
        LessThanOrEqualTo,
        NotEqualTo
```

[end]

6. Open the Web or Windows form in **Design** View.
7. From the **View** menu, click **Code**.

8. In the `ConfigureCrystalReports()` method, you now populate the operatorValueList instance with values from the CeComparisonOperator enum. This code is placed in a different location, depending on whether you are building a Web Site or a Windows project.

   In a Web Site, place the DataSource property assignment and the binding within the Not IsPostBack conditional block.

[Visual Basic]

```
operatorValueList.DataSource =
System.Enum.GetValues(GetType(CeComparisonOperator))
operatorValueList.DataBind()
```

[end]

[C#]

```
operatorValueList.DataSource =
System.Enum.GetValues(typeof(CeComparisonOperator));
operatorValueList.DataBind();
```

[end]

   In a Windows project, place only the DataSource property assignment within the useDefaultValues conditional block.

[Visual Basic]

```
operatorValueList.DataSource =
System.Enum.GetValues(GetType(CeComparisonOperator))
```

[end]

[C#]

```
operatorValueList.DataSource =
System.Enum.GetValues(typeof(CeComparisonOperator));
```

[end]

   Note In a Windows project, you do not need to call a DataBind() method.

9. At the bottom of the `ConfigureCrystalReports()` method, assign the selectionFormula string to the Text property of the formula Label control.

[Visual Basic]

```
formula.Text = selectionFormula
```

[end]

[C#]

```
formula.Text = selectionFormula;
```

[end]

10. Compile and view the application.

    The operator list and all other controls should be correctly displayed.

11. Close your browser.

In the next section, you create a method that retrieves selections from this control when the Redisplay Report button is clicked.

Retrieving Selections from the Control

In this section, you retrieve selections from the DropDownList (Web) or ComboBox (Windows) control.

To retrieve selections from the control

1. Open the Web or Windows form in **Design** view.
2. From the **View** menu, click **Code**.
3. At the bottom of the class, create a private helper method GetSelectedOperator() that returns a string value.

[Visual Basic]

```
Private Function GetSelectedOperator() As String

End Function
```

[end]

[C#]

```
private string GetSelectedOperator()

{

}
```

[end]

4. Within the method, declare a string selectedOperator and assign it an empty string value.

[Visual Basic]

```
Dim selectedOperator As String = ""
```

[end]

[C#]

```
string selectedOperator = "";
```

[end]

5. Still within the method, create a Select Case/switch case statement that checks the selected index in the list control and returns the corresponding enum value.

   Note In C#, you must explicitly cast the selected index to the enum in the switch signature.

[Visual Basic]

```
Select Case operatorValueList.SelectedIndex
    Case CeComparisonOperator.EqualTo
        selectedOperator = "="
    Case CeComparisonOperator.GreaterThan
        selectedOperator = ">"
    Case CeComparisonOperator.GreaterThanOrEqualTo
        selectedOperator = ">="
    Case CeComparisonOperator.LessThan
        selectedOperator = "<"
```

```
        Case CeComparisonOperator.LessThanOrEqualTo
            selectedOperator = "<="
        Case CeComparisonOperator.NotEqualTo
            selectedOperator = "<>"
    End Select
```

[end]

[C#]

```
    switch ((CeComparisonOperator)operatorValueList.SelectedIndex)
    {
        case CeComparisonOperator.EqualTo:
            selectedOperator = "=";
            break;
        case CeComparisonOperator.GreaterThan:
            selectedOperator = ">";
            break;
        case CeComparisonOperator.GreaterThanOrEqualTo:
            selectedOperator = ">=";
            break;
        case CeComparisonOperator.LessThan:
            selectedOperator = "<";
            break;
        case CeComparisonOperator.LessThanOrEqualTo:
            selectedOperator = "<=";
            break;
        case CeComparisonOperator.NotEqualTo:
            selectedOperator = "<>";
            break;
    }
```

[end]

6. Return the selectedOperator from the method.

[Visual Basic]

```
    Return selectedOperator
```

[end]

[C#]

```
    return selectedOperator;
```

[end]

7. From the **File** menu, click **Save All**.

In the last section, you add a button click event to apply the new selection formula values to the report when the button is clicked.

Adding a Button Click Event

The final step in configuring dynamic filtering is to add the button click event and then write code to apply the new selection formula values to the report.

To add the button click event

1. Open the Web or Windows form in **Design** view.
2. Double-click the **Button** control.
3. The code-behind class loads and a button click event is created.
4. Within the method, set the class-level variable "salesAmount" to the Text property of the lastYearsSales control instance.

[Visual Basic]

```
salesAmount = lastYearsSales.Text
```

[end]

[C#]

```
salesAmount = lastYearsSales.Text;
```

[end]

5. Set the class-level variable "operatorValue" to the return value of the GetSelectedOperator helper method.

[Visual Basic]

```
operatorValue = GetSelectedOperator()
```

[end]

[C#]

```
operatorValue = GetSelectedOperator();
```

[end]

6. Set the class-level variable "customerName" to the Text property of the letterOfName control instance.

[Visual Basic]

```
customerName = letterOfName.Text
```

[end]

[C#]

```
customerName = letterOfName.Text;
```

[end]

7. If you are building a Windows project, set the class-level variable "useDefaultValues" to False.

   Note This causes the conditional block within ConfigureCrystalReports() that sets default values for the class-level variables to be skipped.

[Visual Basic]

```
useDefaultValues = False
```

[end]

[C#]

```
    useDefaultValues = false;
[end]
```

8. Still within the method, call the `ConfigureCrystalReports()` method to perform the filtering and binding of the report.

```
[Visual Basic]
    ConfigureCrystalReports()
[end]
[C#]
    ConfigureCrystalReports();
[end]
```

You are now ready to test the application.

Testing the Filtering of Data

In the final section, you test the filtering of data

To test the filtering of data

1. Compile and view your application.
2. Verify that the report is displayed with the default filtering values.
3. Enter "200000" into **last year's sales**.
4. Select **GreaterThan** from the **DropDownList** (Web) or **ComboBox** (Windows).
5. Enter "M" in the **TextBox**, and then click **Redisplay Report**.

   The report redisplays with the data filtered as specified above.
6. Close your browser.

Conclusion

In this tutorial, you filtered data using the RecordSelectionFormula property. You added controls to modify the formula and enable you to filter the data dynamically.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The files for each sample code version are stored in the following folders:

C# Web Site: CS_Web_RDObjMod_FilteringData

C# Windows Project: CS_Win_RDObjMod_FilteringData

Visual Basic Web Site: VB_Web_RDObjMod_FilteringData

Visual Basic Windows Project: VB_Win_RDObjMod_FilteringData

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

ReportDocument Object Model Tutorial:

Displaying Report Parts with the CrystalReportPartsViewer Control

Displaying Report Parts with the CrystalReportPartsViewer Control

Introduction

In this tutorial, you learn how to break a report into parts and display each part by clicking through a series of successive hyperlinks. The hyperlinks move from a general report summary group to increasingly detailed report information. You set the general starting point, and then each successive hyperlink for the report. To display the report as parts, you assign the report to an alternate control named the CrystalReportPartsViewer control.

Note Report parts are used only in Web Sites; they are not available for Windows projects.

There are two primary ways to view a report:

You can view the report as a whole.

You can view the report as a linked series of report parts.

Viewing the report as a whole

Reports are typically displayed as a single document (of one or more pages in length) in which detailed information is shown grouped by categories, subcategories and finally individual rows of detail.

For example, the Xtreme Access database that ships with Crystal Reports contains customer data about bicycle shops from many locations in the world. A typical way to view data in a report is to see the individual bicycle shops that are organized into groups based on regions, and then cities, with individual rows of information for each bicycle shop shown at the detail level.

The CrystalReportViewer control is used to display reports as a whole.

Viewing the report as a linked series of parts

With the introduction of web portals and cell phone web browsing, both of which provide a very limited viewing area for looking at information, a new way of displaying reports has been developed.

To achieve this, the all-in-one report that consisted of group categories, subcategories, and detail rows is redefined as a series of report parts. These parts are then displayed across a series of linked page views. In this scenario, the highest category group of information is displayed first. When an item from that top-level group is selected, a subcategory of group information is displayed. This drill down continues until the individual rows of detail for a particular subcategory are reached.

For example, the Customer data on bicycle shops in the Xtreme Access database is viewed initially as a list of regions. When one region is selected, the cities within that region are displayed. When a particular city is selected, the detail rows of bicycle shops within that particular city are displayed.

This breaking of a report into parts allows a large and complex report to be effectively accessed within a very limited viewing area.

The CrystalReportPartsViewer control is used to display reports as a series of linked parts.

Configuring a report to work with the CrystalReportPartsViewer control

In this tutorial you learn how to take a report that is designed for viewing as a whole and add into it hyperlink information that will enable the report to be accessed as a series of linked parts.

You begin by creating a standard project with a CrystalReportViewer control. You create a report based on the Customer table of the Xtreme sample database and display it in the CrystalReportViewer control as a whole report.

You then re-open the report and access the report parts settings to assign an initial point of entry into the report. (This is typically the highest group category.) You add in hyperlinks that create a linked trail through the report group subcategories until the detail row level has been reached.

This sets the report to be viewed as report parts, but currently it displays as a whole report within the CrystalReportViewer control on the Web Form. To view the report parts, you must delete the CrystalReportViewer control, and then replace it with the CrystalReportPartsViewer control on the Web Form. You update the control variable name in the code-behind class. When you rerun the project, the report is now displayed as a series of linked parts.

Creating a Report

To begin, create a report that draws its information from the sample Microsoft Access database that ships with Crystal Reports.

Note Xtreme.mdb is the sample database that is provided with Crystal Reports. To locate xtreme.mdb on your hard drive for your version of Crystal Reports, see Location of Xtreme Sample Database. You need to connect to the database through its ODBC DSN entry. To learn the name of this entry for your version of Crystal Reports, see ODBC DSN Entry for Xtreme Sample Database.

To create a report with parameters

Note This procedure works only with a project that has been created from Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the template named "Crystal Report."
3. In the **Name** field, enter the name "Customers.rpt" and click **Add**.

   Note In Visual Studio .NET 2002 or 2003, the button is named Open.
4. If you have not registered before, you are asked to register. To find out how to register, see Crystal Reports Registration and Keycode.
5. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
6. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.

7. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to Crystal Reports Developer, the Create New Connection folder does not exist; the contents are shown at the top level.

8. From the subfolder that opens, expand the **ODBC (RDO)** folder.
9. In the ODBC (RDO) window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.

10. Expand the **Tables** node and select the **Customer** table.
11. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
12. CTRL-click **Country**, **Customer Name**, **Last Year's Sales**, **Region** and **City**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click the **Next** button.
14. In the **Available Fields** panel, under **Report Fields**, complete the following instructions:
    a) Select **Country**, click the **>** symbol to move the field into the **Group By** panel.
    b) Select **Region**, click the **>** symbol to move the field into the **Group By** panel.
    c) Click **Next**.

    Summarized fields are created for the Last Year's Sales field in the Country and Region groups.

15. Click **Finish**.

    The Customers report is created and loaded into the main window of Visual Studio.

16. At the bottom of the window, click **Main Report Preview**.

    The report is displayed hierarchically in levels that are sorted first by country and then by region. If you change the report to display as report parts, each level is displayed separately.

You are now ready to convert the report to display report parts.

Converting the Report to Display Report Parts

In this section, you learn how to set hyperlinks in the report to link from general to more detailed information.

To convert the report to display report parts

1. At the bottom of the window, click **Main Report**.

   Note The highest group, **Group #1 Name**, is displayed in two locations, at the top and at the bottom of the report.

   At the bottom of the report in the Group Footer #1 section, **Group #1 Name** is a label that is followed by a summary field for **Customer.Last Year's Sales**. This label is the preferred one to use as a report parts entry point.

2. Right-click **Group #1 Name** that is located in the **Group Footer #1** section, and then click **Copy**.

3. Right-click in an empty area of the report, point to **Report**, and then click **Report Options**.
4. In the Report Options dialog box, click **Paste Link**.

   The ID for the Group 1 label "GroupNameCountry2" is pasted into the **Object Name** field.

   The **Data Context** field accepts a string value that filters report data. For now, this field should remain blank.
5. Click **OK**.
6. Right-click **Group #1 Name**, which is in the **Group Footer #1** section, and then click **Format Object**.
7. In the **Format Editor**, click the **Hyperlink** tab, and then select **Report Part Drilldown**.
8. In the **Available Fields** panel, expand **Group Footer #2**, double-click **GroupNameRegion2**, and then click **OK**.

   You have set Group #1 to link to Group #2 when Group #1 is clicked at runtime.
9. Right-click **Group #2 Name**, which is located in the Group Footer #2 section, and then click **Format Object**.
10. In the **Format Editor**, click the **Hyperlink** tab, and then select **Report Part Drilldown**.
11. In the **Available Fields** panel, do the following:
    b) Expand **Details**.
    d) CTRL-click **CustomerName1**, **LastYearsSales1**, and **City1**.
    e) Click the > symbol to move these fields into the **Fields to Display** panel.
    f) Click **OK**.

       You have set Group #2 to link to detailed information about the customer, when Group #2 is clicked at runtime.
12. From the **File** menu, select **Save All**.

Binding the Report

When you followed the instructions in the section Appendix: Project Setup to prepare for this tutorial, you placed a CrystalReportViewer control on the Web Form. However, to view report parts, you must use the CrystalReportPartsViewer control.

In this section, you instantiate the Customers report and bind it to the CrystalReportPartsViewer control. Then you test whether the report displays the report parts that you have created in the previous procedures.

You can instantiate and bind the report two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the report as a non-embedded report.

To instantiate the Customers report as an embedded report and bind it to the CrystalReportPartsViewer control

1. Open the Web Form.
2. From the **View** menu, click **Designer**.
3. Delete the **CrystalReportViewer** control.
4. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportPartsViewer** control.
5. Drag the **CrystalReportPartsViewer** control onto the Web Form.
6. From the **Properties** window, set the **ID** property:

   For Visual Basic Web Sites, set the ID property to "myCrystalReportPartsViewer"

   For C# Web Sites, set the ID property to "crystalReportPartsViewer".
7. From the **View** menu, click **Code** to view the code-behind class for this Web Form.
8. Add a new class-level declaration for the Customers report wrapper class, with the variable name customersReport. Set its access modifier to private.

[Visual Basic]

```
Private customersReport As Customers
```

[end]

[C#]

```
private Customers customersReport;
```

[end]

9. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
customersReport = New Customers()
```

[end]

[C#]

```
customersReport = new Customers();
```

[end]

10. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportPartsViewer control to the instantiated report class (variable name: customersReport).

[Visual Basic]

```
myCrystalReportPartsViewer.ReportSource = customersReport
```

[end]

[C#]

```
crystalReportPartsViewer.ReportSource = customersReport;
```

[end]

Note The CrystalReportPartsViewer control instance is accessible in the code because you added the control to your Web or Windows form. If IntelliSense does not recognize the CrystalReportPartsViewer control instance, verify that the CrystalReportPartsViewer control has been added as a class-level declaration to this code-behind class.

You are now ready to build and run your project.

To instantiate the Customers report as a non-embedded report and bind it to the CrystalReportPartsViewer control

1. Open the Web Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, with the variable name customersReport. Set its access modifier to private.

[Visual Basic]

```
Private customersReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument customersReport;
```

[end]

Note The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (which you added during one of the procedures in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
customersReport = New ReportDocument()
```

[end]

[C#]

```
customersReport = new ReportDocument();
```

[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("Customers.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("Customers.rpt");
```

[end]

For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"Customers.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" + "Customers.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
customersReport.Load(reportPath)
```

[end]

[C#]

```
customersReport.Load(reportPath);
```

[end]

7. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportPartsViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportPartsViewer.ReportSource = customersReport
```

[end]

[C#]

```
crystalReportPartsViewer.ReportSource = customersReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: customersReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project. It is expected that the report loading will fail, because code has not yet been written to set a value for the City parameter field. You'll add a value for the City parameter field later in this tutorial.

To test the loading of the Customers report

1. From the **Build** menu select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The Customers report is displayed in the Web browser.

4. Click a country to display a list of regions.
5. Click a region to display the Customer Name, Last Year's Sales, and City fields.
6. Return to Visual Studio and click **Stop** to exit from debug mode.

Filtering Data in the Report

In this section, you learn how to set the Data Context field to filter data that is displayed on the report.

To filter data in the report

1. Open the Crystal report.
2. Right-click in an empty area of the report. Select **Report**, and then click **Report Options**.
3. In the Data Context field, type "/Country[USA]".
4. Click **OK** to close the **Report Options** dialog.
5. From the **Build** menu, select **Build Solution**.
6. If you have any build errors, go ahead and fix them now.
7. From the **Debug** menu, click **Start**.

   The report displays only records from the USA.
8. Return to Visual Studio and click **Stop** to exit from debug mode.

You can choose to remove or modify the Data Context settings.

Conclusion

In this tutorial, you learned how to break a report into parts and display each part by clicking through a series of successive hyperlinks. The hyperlinks moved from a general report summary group, to increasingly detailed report information. To view the report as parts, you assigned the report to the CrystalReportPartsViewer control.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project, or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_RDObjMod_ReportParts

Visual Basic Web Site: VB_Web_RDObjMod_ReportParts

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorials in Visual Studio 2005

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorial:

Web Site Setup with Crystal Reports Using Smart Tasks

Reduced-Code Web Site Setup with Crystal Reports Using Smart Tasks

Introduction

In this tutorial, you learn how to use the new tag-based application development model in Visual Studio 2005. You follow a reduced-code development model to set up a Web Site with Crystal Reports, with Smart Tasks.

In this tutorial, you create an ASP.NET Web Site in Visual Studio 2005. Then you add a CrystalReportViewer control to the Web Form, which launches the new Smart Task panel. From Smart Tasks, you create a CrystalReportSource control, and then build a Crystal report. The report is referenced within the CrystalReportSource control.

Throughout this tutorial, you can preview the report three ways:

In the embedded Crystal Reports Designer, with the new Preview button in Crystal Reports for Visual Studio 2005.

In the Web Form, with dummy data.

Note This preview of the Web Form is the new Design Time Preview feature. For more information, see Appendix: Design Time Preview.

At runtime, when you build and compile the application.

At the end of this tutorial, you switch to HTML view (Source View) and explore the relationship between the CrystalReportViewer control and the CrystalReportSource control, in the new tag-based application model of Visual Studio 2005.

Creating a Web Site with a CrystalReportViewer Control

Before you create a Web Site, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a reduced-code Web Site in Crystal Reports for Visual Studio 2005

1. Start Visual Studio 2005.
2. From the **File** menu, click **New**, and then select **Web Site**.
3. In the **New Web Site** dialog box, click **ASP.NET Web Site**.
4. In the **Location** dropdown, select **File System**.
5. In the **Language** dropdown, select the coding language that you wish to use.
6. In the **Location** text field enter the directory path "C:\WebSites\", followed by the name of your project.

   ```
   C:\WebSites\MyProjectName
   ```
7. Click **OK**.

   Your project opens in Solution Explorer and contains a Default.aspx page.
8. Open the Default.aspx page (the Web Form).
9. From the **View** menu, click **Designer**.

   Note You can also switch to design view by clicking the Design button at the bottom of the form view.

10. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
11. Drag the **CrystalReportViewer** control onto the Web Form.

    The "CrystalReportViewer Tasks" Smart Task panel opens.

Configuring the CrystalReportSource in Smart Tasks

Visual Studio 2005 has a new GUI feature for .NET controls that is called Smart Tasks. For the CrystalReportViewer control in Web Sites, the Smart Task panel is named "CrystalReportViewer Tasks."

The "CrystalReportViewer Tasks" Smart Task panel enables you to configure several features of the CrystalReportViewer control, without having to write code. Any selections that you make through the use of Smart Tasks are generated as tag-based settings within the ASPX page.

In this section, you learn how to configure the CrystalReportSource option that is available from the "CrystalReportViewer Tasks" Smart Task panel.

Later in these reduced-code tutorials, you examine other features of the "CrystalReportViewer Tasks" Smart Task.

To begin, you learn how to close and re-open Smart Tasks.

To configure the CrystalReportSource using Smart Tasks

1. If the Smart Task panel is currently open, click the Web Form to close the Smart Task panel.
2. On the upper-right corner of the CrystalReportViewer control, click the small triangular button.

   The Smart Task panel named "CrystalReportViewer Tasks" re-opens. In the Choose Report Source list, <None> is displayed.
3. Click the **Choose Report Source** list and select **<New report source...>**.

   The Create ReportSource dialog box opens.
4. In the **Specify a name for the control** textbox, leave the default entry as CrystalReportSource**1**.
5. Click the **Specify a report for the CrystalReportSource control** list and select **<New Report>**.

   Your alternative selection is <Browse…>. That option allows you to select an existing Crystal report from the file directory. However, in this tutorial you create a new report.
6. In the **Create a New Crystal Report** dialog box, enter the name "XtremeCustomers.rpt," and then click **OK**.
7. Click **OK** again to close the **Choose Report Source** dialog box.

Configuring the Report for the CrystalReportSource Control

In this section, you use the Report Wizard to configure the report and add it to the CrystalReportSource control. You create a database connection, set up the fields for the

report, and then add the report to the control. Once you are finished, you test your reduced-code Web Site.

To configure the report for the CrystalReportSource control

1. If the Registration Wizard dialog box appears, you need to register your copy of Crystal Reports. You have two choices:

   Click **Next** to register now.

   Click **Register Later** to register later.

   After registering or choosing to register later, proceed to the next step.
2. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
3. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
4. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, do the following:

   f) Expand the **Create New Connection** folder.

   g) Expand the **ODBC (RDO)** folder.
5. In the **ODBC (RDO)** window, select the correct ODBC DSN entry for your version of Crystal Reports, as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
6. Expand the **Tables** node and select the **Customer** table.
7. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
8. Expand the **Customer** table, then CTRL-click **Customer Name**, **Contact Title**, **Address1**, **Contact Last Name** and **City**.
9. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Next**.
10. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

    The XtremeCustomers report is created and loaded into the main window of Visual Studio.
11. At the bottom of the window, click **Main Report Preview**.

    The report is displayed in preview mode and shows data from the Xtreme database.

    Note The Preview button is available for the first time in the embedded Crystal Reports Designer that ships with Crystal Reports for Visual Studio 2005.
12. At the bottom of the window, click **Main Report** to exit preview mode.
13. Open the Web Form.
14. From the **View** menu, click **Designer**.

    The report is displayed in Design Time Preview mode, which shows the graphic layout for the report on the Web Form.
15. From the **File** menu, click **Save All**.

Your report is created and designated as the report for the CrystalReportSource control. You are now ready to build and run your project.

To test the reduced-code Web Site

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.
4. The XtremeCustomers report is displayed in the Web browser.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Exploring the CrystalReportSource Control in Source View

In this section, you explore the tag-based application model that is used in ASP.NET version 2.0. This model makes it possible for you to set up a Web Site with Crystal Reports for Visual Studio 2005, without having to write code. You also learn about the DataSource control framework and the CrystalReportSource control that are part of ASP.NET 2.0.

To explore the CrystalReportSource control in Source view

1. Open the **Default.aspx** page.
2. From the **View** menu, click **Designer**.
3. At the bottom of the Web Form, click the **Source** button.

   The HTML for the Default.aspx page is displayed. This includes two related tags for Crystal Reports:

   The CrystalReportSource tag, which nests the Report tag. The Report tag references the report in its FileName property.

```
<CR:CrystalReportSource ID="CrystalReportSource1" Runat="server"
EnableCaching="False" GroupPath="">
    <Report FileName="XtremeCustomers.rpt"></Report>
</CR:CrystalReportSource>
```

   The CrystalReportViewer tag, which references the CrystalReportSource by ID.

```
<CR:CrystalReportViewer ID="CrystalReportViewer1" Runat="server"
AutoDataBind="True"
    Height="1058px" CssFilename="CssFilename"
ReportSourceID="CrystalReportSource1" Width="919px" />
```

The code demonstrates the new tag-based application model that is part of ASP.NET 2.0. Similar to ASP.NET version 1.0, the CrystalReportViewer control is represented by an XML tag. However, the available properties for this control have been increased in ASP.NET 2.0, to support the tag-based application development model.

In ASP.NET 2.0, data connectivity has been encapsulated within the DataSource control. The CrystalReportSource control adapts the DataSource control framework, except that it encapsulates report connectivity, rather than data connectivity. This framework is handled by the ReportSourceID property, in the main visual control.

The properties are also demonstrated in other reduced-code tutorials later in this section.

Conclusion

In this tutorial, you successfully created a Crystal Reports Web Site through use of the reduced-code development model. You learned to add a CrystalReportViewer control to the Web Form, to use Smart Tasks to create a CrystalReportSource control, and to assign a Crystal report to the control.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_ReducedCode_WebSiteSetup

Visual Basic Web Site: VB_Web_ReducedCode_WebSiteSetup

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorial:

Windows Project Setup with Crystal Reports Using Smart Tasks

Reduced-Code Windows Project Setup with Crystal Reports Using Smart Tasks

In this tutorial, you learn how to follow a reduced-code development model to set up a Windows project, using Smart Tasks.

Introduction

In this tutorial, you learn how to create a Windows project in Crystal Reports for Visual Studio 2005. You then add a CrystalReportViewer control to the Windows Form. From the Smart Task panel, you open the embedded Crystal Reports Designer and create a new report. The Smart Tasks feature generates code that binds the report to the CrystalReportViewer control.

Throughout this tutorial, you can preview the report three ways:

In the embedded Crystal Reports Designer, with the new Preview button in Crystal Reports for Visual Studio 2005.

In the Windows Form, with dummy data.

Note This preview of the Windows Form is a new feature called the Design Time Preview. For more information, see Appendix: Design Time Preview.

At runtime, when you build and compile the application.

At the end of this tutorial, you switch to code view, to see how the reduced-code development model functions in Crystal Reports for Visual Studio 2005.

Creating a Windows Project with a CrystalReportViewer Control

Before you create a Windows Project, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a reduced-code Windows project in Crystal Reports for Visual Studio 2005

1. Launch Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Project**.
3. In the **New Project** dialog box, select a language folder for C# or Visual Basic from the **Project Types** list.
4. From the **Templates** list, click **Windows Application**.
5. In the **Name** field, replace the default project name with the name of your project.
6. Click **OK**.

   Your project opens in Solution Explorer and contains a Form1 class.
7. Open the Form1 class.
8. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
9. Drag the **CrystalReportViewer** control onto the Windows Form.
10. From the **Properties** window, set the **Name** property:

    For Visual Basic projects, set the value to "myCrystalReportViewer."

For C# projects, set the value to "crystalReportViewer."

11. From the **File** menu, click **Save All**.

Configuring the Report in Smart Tasks

Visual Studio 2005 has a new GUI feature for .NET controls, called Smart Tasks. For the CrystalReportViewer control in Windows projects, the Smart Task panel is named "CrystalReportViewer Tasks."

The "CrystalReportViewer Tasks" Smart Task panel enables you to configure several features of the CrystalReportViewer control, without having to write code.

Any selections that you make in Smart Tasks are generated as code in a Form1 partial class. This hidden partial class is a new feature of Visual Studio 2005 that contains initialization and design code. In this section, you configure a new report with the CrystalReportViewer Tasks Smart Task panel and the embedded Crystal Reports Designer. Later in these reduced-code tutorials, you examine other features of Smart Tasks.

Configuring the Report using Smart Tasks

1. On the upper-right corner of the CrystalReportViewer control, click on the small triangular button.

   The Smart Task panel named "CrystalReportViewer Tasks" opens.
2. Click the **Create a New Report** link.
3. In the **Create a New Crystal Report** dialog box, enter the name "XtremeCustomers.rpt," and then click **OK**.

   The Crystal Reports Gallery dialog box opens.
4. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
5. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
6. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, do the following:

   a) Expand the **Create New Connection** folder.

   b) Expand the **ODBC (RDO)** folder.
7. In the **ODBC (RDO)** window, select the correct ODBC DSN entry for your version of Crystal Reports, as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
8. Expand the **Tables** node, and then select the **Customer** table.
9. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
10. CTRL-click **Customer Name**, **Contact Title**, **Address1**, and **City**.
11. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Next**.
12. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

The XtremeCustomers report is created and loaded into the main window of Visual Studio.

13. At the bottom of the window, click **Main Report Preview**.

    The report is displayed in preview mode and shows data from the Xtreme database.

    Note The Preview button is a new feature of the embedded Crystal Reports Designer that ships with Crystal Reports for Visual Studio 2005.

14. At the bottom of the window, click **Main Report** to exit Preview mode.
15. Return to the **Form1** class.

    The report is displayed in Design Time Preview mode, showing the report's graphic layout on the Web Form. Below Form1, the XtremeCustomers report is displayed in the component tray.

16. From the **File** menu, click **Save All**.

Your report is created and becomes the designated report for the CrystalReportViewer control. At this point, you may want to rename your report variable.

To rename the report variable

1. Click **XtremeCustomers1** in the component tray to select it.
2. From the **Properties** window, set the **Name** field to "xtremeCustomersReport."

You are now ready to build and run your project.

To test the reduced-code Windows project

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.
4. The XtremeCustomers report is displayed in a new Windows application.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Exploring the Code Generated by Smart Tasks

In this section, you explore the code that is generated by Smart Tasks, which enabled you to set up a Windows Project with Crystal Reports without having to write code.

To explore the code generated by Smart Tasks

1. Open the **Form1** class.
2. From the **View** menu, click **Code**.

   The Form1 class opens in code view and displays an empty class.

   Note In C#, the empty class displays the class constructor.

```
public Form1()
{
    InitializeComponent();
}
```

Notice that the following code is missing:

The declaration of the CrystalReportViewer control instance

The instance of the XtremeCustomers report.

The code that binds the report to the CrystalReportViewer control.

The absent code has been hidden in a partial class. Partial classes are a new feature in the .NET Framework 2.0 and Visual Studio 2005. Partial classes allow one class to accept an addendum from another class file, by giving the other class file the same class name and prefixed with the word "partial." The partial class usually contains auto-generated designer code.

To view the auto-generated code that does not appear in the Form1 class, you will explore the CrystalReportViewer and report instances in the hidden Form1 partial class.

To view the hidden Form1 partial class

1. From **Solution Explorer**, click the toolbar icon **Show All Files**.
2. Expand **Form1.cs** or **Form1.vb**.

   The Form1.Designer.cs or .vb is displayed.
3. Right-click **Form1.Designer.cs** or **.vb** and select **View Code**.

   The Form1 partial class opens and displays the auto-generated code for this class. This includes the declaration of the CrystalReportViewer control instance, the XtremeCustomers report instance, and the code that binds the report to the CrystalReportViewer control instance.

   Note If you prefer to write the code and place it into the code-behind class, see Appendix: Project Setup.

Conclusion

In this tutorial, you followed a reduced-code development model to create a Crystal Reports Windows Project.

You added a CrystalReportViewer control, and then used Smart Tasks to create a new report and bind it to the CrystalReportViewer control, without having to write code. You then explored the code that was auto-generated in the hidden partial class, based on your selections in Smart Tasks.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Windows project: CS_Win_ReducedCode_ProjectSetup

Visual Basic Windows project: VB_Win_ReducedCode_ProjectSetup

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorial:

Secure Database Logon in a Web Site

Reduced-Code Secure Database Logon in a Web Site

Introduction

In this reduced-code Web Site tutorial, you learn how to display a report that contains information from a secure SQL Server database. To provide a secure database logon, you assign the report's DataSources property to the SQLDataSource control, which is part of the new DataSource control framework in Visual Studio 2005.

> **Note** A coding development model of this tutorial is also available. See Logging onto a Secure SQL Server Database Using SQL Authentication or Logging onto a Secure SQL Server Database Using Integrated Security.

In this tutorial, to display a report that contains information from a secure SQL Server database, you use the new tag-based application development model to connect the key components:

Connect a CrystalReportViewer control to a CrystalReportSource control.

Connect the CrystalReportSource control to a report.

Connect the report's DataSources property to an SqlDataSource control.

> **Note** DataSources is viewed as a report property in the Properties window, or as a tag in HTML view (Source view).

All of this work is done without having to write code.

To begin, you add a CrystalReportViewer control to the Web Form. From Smart Tasks, you create a CrystalReportSource control, and then build a Crystal report with the embedded Crystal Reports Designer. This report logs on to a secure SQL server database. The report is referenced within the CrystalReportSource control.

Then you learn how to configure the logon to the secure SQL server database, to return the required tables that are used by the Crystal report. To start, you select the "Enable Database Logon prompt" option in Smart Tasks. When that option is selected, the exception that is thrown during a database logon failure is replaced with a prompt that requests the user to log on manually.

To configure a consistent log on to the required database, you add a SqlDataSource control, and then configure it to address the secure SQL server database and return the required tables.

Finally, you assign the DataSource property of the report to the SqlDataSource control that you have created.

Creating a Web Site with a CrystalReportViewer Control

Before you create a Web Site, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

Some database setup is required as a prerequisite to this tutorial.

Prerequisite Database Setup

1. SQL Server configuration:

If you have SQL Server (or the OEM version, MSDE) installed, it must be configured to require SQL Server Authentication.

If you do not have SQL Server (or the OEM version, MSDE) installed, you must install MSDE with SQL Server Authentication set to "True."

2. The Northwind database provided with SQL Server must be installed and verified that it requires SQL Server Authentication.
3. A limited access account must be created for use within the Web site.

To install MSDE with SQL Server Authentication, or the Northwind database, go to the following sections from Appendix: System Setup in this documentation:

Appendix: MSDE Installation with Windows or SQL Server Authentication

Appendix: Northwind Database Installation

Appendix: Security: Creating a Limited Access Database Account

Once you have configured SQL Server and the Northwind database according to the sections above, you are ready to create a Web Site that displays a Crystal report drawing its information securely from a Northwind database.

You begin by setting up a reduced-code Web Site.

To set up a reduced-code Web Site in Crystal Reports for Visual Studio 2005

1. Launch Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Web Site**.
3. In the **New Web Site** dialog box, click **ASP.NET Web Site**.
4. In the **Location** dropdown, select **File System**.
5. In the **Language** dropdown, select the coding language that you wish to use.
6. In the **Location** text field enter the directory path "C:\WebSites\", followed by the name of your project.

   ```
   C:\WebSites\MyProjectName
   ```

7. Click **OK**.

   Your project opens in the Solution Explorer and contains a Default.aspx page.
8. Open the Default.aspx page (the Web Form).
9. From the **View** menu, click **Designer**.

   Note You can also switch to design view by clicking the Design button at the bottom of the form view.
10. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
11. Drag the **CrystalReportViewer** control onto the Web Form.

    The Smart Task panel named "CrystalReportViewer Tasks" opens.

Configuring the CrystalReportSource in Smart Tasks

Visual Studio 2005 has a new GUI feature for .NET controls, called Smart Tasks. For the CrystalReportViewer control in Web Sites, the Smart Task panel is named "CrystalReportViewer Tasks."

The "CrystalReportViewer Tasks" Smart Task panel enables you to configure several features of the CrystalReportViewer control, without having to write code.

Any selections that you make in Smart Tasks are generated as tag-based settings within the ASPX page.

In this section, you configure the CrystalReportSource control that is available from the "CrystalReportViewer Tasks" Smart Task. Later in these reduced-code tutorials, you examine other features of Smart Tasks.

To begin, you learn how to close and re-open Smart Tasks.

Configuring the CrystalReportSource using Smart Tasks

1. If the Smart Task panel is currently open, click on the Web Form to close Smart Tasks.
2. On the upper-right corner of the CrystalReportViewer control, click on the small triangular button.

   The Smart Task panel named "CrystalReportViewer Tasks" re-opens. In the Choose Report Source list, <None> is displayed.
3. Click the **Choose Report Source** list and select **<New report source...>**.

   The Create ReportSource dialog box opens.
4. In the **Specify a name for the control** textbox, leave the default entry as CrystalReportSource**1**.
5. Click the **Specify a report for the ReportSource control** drop- down list and select **<New Report>**.

   Your alternative selection is <Browse...>. That option allows you to select an existing Crystal report from the file directory. However, later you will create a new report that is required for this tutorial.
6. In the **Create a New Crystal Report** dialog box, enter the name "NorthwindCustomers.rpt".
7. Click **OK**, and then click **OK** again to close the **Create ReportSource** dialog box.

Creating and Connecting the Report to a Secure Database

You now begin the process of creating and connecting the report.

To create and connect the report to a secure database

1. If the Registration Wizard dialog box appears, you need to register your copy of Crystal Reports. You have two choices:

   Click **Next** to register now.

   Click **Register Later** to register later.

   After registering or choosing to register later, proceed to the next step.
2. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
3. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
4. In the **Available Data Sources** panel, expand the **Create New Connection** folder.

Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to Crystal Reports Developer, the Create New Connection folder does not exist; the contents are shown at the top level.

5. From the subfolder that opens, expand the **OLE DB (ADO)** folder.
6. In the **OLE DB (ADO)** dialog box, select **Microsoft OLE DB Provider for SQL Server**, and then click **Next**.
7. You can now use either SQL Authentication, or Windows Authentication with Integrated Security:

   If you are using SQL Authentication, enter the values for your database server, user id, and password into the **Server**, **User ID**, and **Password** fields and then from the **Database** list, select "Northwind."

   Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

   If you are using Windows Authentication with Integrated Security, enter the value for your database server and check the **Integrated Security** checkbox, and then from the **Database** list, select "Northwind."
8. Click **Finish**.

   The OLE DB folder expands to show your database server and, within it, the Northwind database.
9. Expand the nodes **Northwind**, **dbo**, and **Tables**, and then select the **Customers** table.
10. Click the **>** symbol to move the table into the **Selected Tables** panel, and then click **Next**.
11. Hold down CTRL and click **CompanyName**, **ContactName** and **City**.
12. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Next**.
13. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

    The NorthwindCustomers report is created and loaded into the main window of Visual Studio.
14. At the bottom of the window, click **Main Report Preview**.

    The report is displayed in preview mode and shows data from the secure Northwind database.

    Note The Preview button is a new feature of the embedded Crystal Reports Designer that ships with Crystal Reports for Visual Studio 2005.
15. At the bottom of the window, click **Main Report** to exit Preview mode.
16. Open the Web Form.
17. From the **View** menu, click **Designer**.

    The report is displayed in Design Time Preview mode and shows the report's graphic layout on the Web Form.
18. From the **File** menu, click **Save All**.

Your report is created and is now the designated report for the CrystalReportSource control. You are now ready to build and run your project.

To test the reduced-code Web Site

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.
4. A database logon prompt page appears, asking you to enter the database password.

   This is the expected result, for two reasons:

   You have not yet configured the data source.

   In Smart Tasks for the CrystalReportViewer control, the "Enable database logon prompting" checkbox is selected by default.
5. Enter the password and click **Logon**.
6. The NorthwindCustomers report is displayed in the Web browser.
7. Return to Visual Studio and click **Stop** to exit from debug mode.

Configuring Database Logon Prompting in the Smart Task

In this procedure, you examine the effect of turning off "Enable Database Logon Prompting" in the Smart Task.

To configure Database Logon Prompting in the Smart Task

1. Open the Default.aspx page (the Web Form).
2. From the **View** menu, click **Designer**.
3. On the upper-right corner of the CrystalReportViewer control, click on the small triangular button.

   The Smart Task panel "CrystalReportViewer Tasks" opens.
4. Clear the **Enable Database Logon Prompting** checkbox.
5. Click on the Web Form to close the Smart Task.
6. From the **Build** menu, select **Build Solution**.
7. If you have any build errors, go ahead and fix them now.
8. From the **Debug** menu, click **Start**.

   Because database logon prompting is no longer enabled, a Logon Failed exception is thrown in the Web application window.
9. Return to Visual Studio and click **Stop** to exit from debug mode.
10. Return to the Smart Task panel for the CrystalReportViewer control.
11. Select the **Enable Database Logon Prompting** checkbox.
12. Close **Smart Tasks**.
13. From the **File** menu, click **Save All**.

Adding a SqlDataSource control

In ASP.NET 2.0, data sources are now configured and accessed through the new DataSource control framework. By default, a Crystal report bypasses the need for a separate data source because data access is encapsulated within the report.

However, if the data is from a secure SQL server, then the report requires a data source to manage the logon process.

To add a SqlDataSource control

1. Open the Default.aspx page (the Web Form).
2. From the **View** menu, click **Designer**.

   Note You can also switch to design view by clicking the Design button at the bottom of the form view.
3. Click your mouse to the right of the CrystalReportSource control, then press ENTER.

   A flashing cursor appears below the CrystalReportSource control.
4. In the **Toolbox**, click the **Data** sub node to expand the Data controls.
5. From the **Data** sub node, drag a **SqlDataSource** control onto the Web Form below the CrystalReportSource control.
6. In the **Smart Task panel** (named **Common SqlDataSource Tasks**), click **Configure Data Source...**
7. In the **Configure Data Source** window, click **New Connection...**
8. In the **Add Connection** dialog box, in the **Server** name field, enter the name of your database server.
9. In the **Log on to the server** panel, choose either the **Use SQL Server Authentication** or the **Use Windows Authentication** radio button.
10. If you chose **SQL Server Authentication**, do the following:

    Enter values into the **User name** and **Password** fields.

    Note Remember to use the limited access database account.

    Select the **Save my password** checkbox.
11. If you chose **Windows Authentication**, go to the next step.
12. In the **Connect to a database** panel, select the **Select or enter a database name** radio button.
13. From the list, select **Northwind**, and then click **Test Connection**.
14. The Northwind connection is verified. If the connection fails, check your password.
15. Once the test connection succeeds, click **OK** to close the **Connection Properties** window.
16. In the **Configure Data Source** window, the **Connection String** field shows the complete connection information.

    If you chose Windows Authentication, the Integrated Security property is set to true.

    ```
    Data Source=ABCDE;Initial Catalog=Northwind;Integrated Security=True
    ```

    If you chose SQL Authentication, the UserID and password values are assigned.

Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

In the code that you write, replace the sample server name, database name and password (shown below) with your own connection information.

```
Server=ABCDE;User
ID=limitedPermissionAccount;Password=1234;Database=Northwind;Persist
Security Info=True
```

17. Click **Next**, and in the following window click **Next** again to save the connection with a default name of "NorthwindConnectionString."
18. Click the **Specify columns from a table or view** option.
19. In the **Name** list, select "Customers."
20. In the **Columns** pane, select the **Company Name**, **Contact Name** and **City** checkboxes, and then click **Next**.

    Note The columns selected here must match the columns that are selected in your original report. If you later change the columns that are selected in your report, with the embedded Crystal Reports Designer, you must change the columns here in your SQL statement. For scalability purposes, it is strongly recommended that you do not use the asterisk selection, but instead choose the specific columns that are used by your report.
21. In the **Test Query** window, click **Test Query**.
22. If the query is successful, then click **Finish**.
23. The **SqlDataSource** control is displayed at the bottom of the window, below the **CrystalReportViewer** control and the CrystalReportSource control.

Connecting CrystalReportSource to the SqlDataSource Control

On your Web Form, you now have three controls that are related to each other:

The CrystalReportViewer control, which links to the CrystalReportSource control to access the report to display it.

The CrystalReportSource control, which encapsulates the report.

The SqlDataSource control, which encapsulates data access information.

Remember that the CrystalReportSource control was linked to the CrystalReportViewer control when adding the CrystalReportViewer control to the Web Form. However, nothing has occurred to link the CrystalReportSource to the SqlDataSource. Therefore, even though the data configuration information exists, at runtime the report would not be able to find this information and would ask for logon.

In this section, you connect the CrystalReportSource to the SqlDataSource control. For this task, you use the DataSource Collection Editor dialog box.

To connect the CrystalReportSource to the SqlDataSource control

1. Open the Default.aspx page (the Web Form).
2. From the **View** menu, click **Designer**.

3. Select the **CrystalReportSource** control.
4. From **Properties**, expand **Report**.

   Note If the Properties Explorer is not visible, from the **View** menu, click **Properties Window**.
5. Within the **Report** property, click the ellipsis (**...**) on the far right of **DataSources (Collection)**.
6. In the **Data Source Collection Editor** dialog box, do the following:
   a) Click **Add**.
   b) In the **Report** panel, select **Main report**.
   c) In the **Choose Data Source** combo box, click **SqlDataSource1**.
   d) Select **Specify Table Name**, and in the field type "Customers".
7. Click **OK**.
8. At the bottom of the Default.aspx page, click **Source**.
9. Locate the CrystalReportSource tags.
10. Within the CrystalReportSource tags, locate the Report tag.
11. Within the Report tags (between the opening and closing tags), verify that the following DataSources tagging information has been generated.

```
<DataSources>
    <CR:DataSourceRef DataSourceID="SqlDataSource1"
TableName="Customers" />
</DataSources>
```

12. From the **Build** menu, click **Build Solution**.
13. If you have any build errors, go ahead and fix them now.
14. From the **Debug** menu, click **Start**.

    The report is now successfully displayed with secure logon to the SQL Server database.

Conclusion

In this reduced-code Web Site tutorial, you created and displayed a report that contained information from a secure SQL Server database. You enabled and disabled a database logon prompt, by configuring the Smart Task. You then added a SqlDataSource control to the Web Form and configured it to access the secure SQL Server database. Then, to connect the SqlDataSource to the report, you assigned the SqlDataSource ID to the DataSource tag, within the Report tag.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_ReducedCode_DBLogon

Visual Basic Web Site: VB_Web_ReducedCode_DBLogon

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorial:

Parameter Setting in a Web Site

Reduced-Code Parameter Setting in a Web Site

Introduction

In this reduced-code Web Site tutorial, you learn how to display a report that contains parameters, and how to pass parameter values to that report with the new Data Parameters model.

> **Note** A coding model of this tutorial is also available. See Reading and Setting Discrete Parameters.

In this tutorial, you use the new tag-based application development model to connect the key components:

- Connect a CrystalReportViewer control to a CrystalReportSource control.
- Connect the CrystalReportSource control to a report that contains a parameter.
- Connect the report's Parameters tag to one of the parameter sources in the new Data Parameters model, which is included with ASP.NET version 2.0.

All of this work is done without having to write code.

To begin, you add a CrystalReportViewer control to the Web Form. From Smart Tasks, you create a CrystalReportSource control, and then build a Crystal report with the embedded Crystal Reports Designer. This report includes a parameter that is applied to search criteria for the report. The report is referenced within the CrystalReportSource control.

You then use the Enable Parameters Prompt option in Smart Tasks. When that option is enabled, the exception that is thrown during a database logon failure is replaced with a prompt that requests the user to supply a parameter.

However, you need a means to customize your parameter selection. To do that, you add a DropDownList that contains a list of parameter values that can be selected for the report.

Finally, you open the Parameter Collection Editor for the report, and configure a connection between the report parameter and the DropDownList control that you have added.

> **Note** A ControlParameter is only one possible parameter source in the Data Parameters model. Other possible parameter sources include the QueryStringParameter, SessionParameter, FormParameter or CookieParameter. For some of these parameter source types you edit the Parameters tag directly in HTML view (source view).

Creating a Web Site with a CrystalReportViewer control

Before you create a Web Site, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a reduced-code Web Site in Crystal Reports for Visual Studio 2005

1. Start Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Web Site**.

3. In the **New Web Site** dialog box, click **ASP.NET Web Site**.
4. In the **Location** dropdown, select **File System**.
5. In the **Language** dropdown, select the coding language that you wish to use.
6. In the **Location** text field enter the directory path "C:\WebSites\", followed by the name of your project.

   ```
   C:\WebSites\MyProjectName
   ```

7. Click **OK**.

   Your project is displayed in the Solution Explorer and contains a Default.aspx page.
8. Open the Default.aspx page (the Web Form).
9. From the **View** menu, click **Designer**.

   Note You can also switch to design view by clicking the Design button at the bottom of the form view.
10. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
11. Drag the **CrystalReportViewer** control onto the Web Form.

    The "CrystalReportViewer Tasks" Smart Task panel opens.

Configuring the CrystalReportSource in Smart Tasks

Visual Studio 2005 has a new GUI feature for .NET controls, called Smart Tasks. For the CrystalReportViewer control in Web Sites, the Smart Task panel is named "CrystalReportViewer Tasks."

The "CrystalReportViewer Tasks" Smart Task panel enables you to configure several features of the CrystalReportViewer control, without having to write code. Any selections that you make in Smart Tasks are generated as tag-based settings within the ASPX page.

In this section, you configure the main feature that is available from the "CrystalReportViewer Tasks" Smart Task: the CrystalReportSource control. Later in these reduced-code tutorials, you examine other features of Smart Tasks.

To begin, you learn how to close and re-open Smart Tasks.

To configure the CrystalReportSource using Smart Tasks

1. If the Smart Task panel is currently open, click on the Web Form to close the Smart Task panel.
2. On the upper-right corner of the CrystalReportViewer control, click on the small triangular button.

   The named "CrystalReportViewer Tasks" re-opens. In the Choose Report Source list, <None> is displayed.
3. Click the **Choose Report Source** list and select **<New report source...>**.

   The Create ReportSource dialog box opens.
4. In the **Specify a name for the control** textbox, leave the default entry as **CrystalReportSource1**.
5. Click the **Specify a report for the ReportSource control** drop- down list and select **<New Report>**.

Your alternative selection is <Browse…>. That option allows you to select an existing Crystal report from the file directory. However, later you will create a new report that is required for this tutorial.

6. In the **Create a New Crystal Report** dialog box, enter the name "XtremeCustomers.rpt" and click **OK**, and then click **OK** again within the Create ReportSource dialog box.

Creating the Report with Parameters

To begin, create a report that draws its information from the sample Microsoft Access database that ships with Crystal Reports.

Note Xtreme.mdb is the sample database that is provided with Crystal Reports. To locate xtreme.mdb on your hard drive for your version of Crystal Reports, see Appendix: Location of Xtreme Sample Database. You need to connect to the database through its ODBC DSN entry. To learn the name of this entry for your version of Crystal Reports, see Appendix: ODBC DSN Entry for Xtreme Sample Database.

To create the report with parameters

1. If the Registration Wizard dialog box appears, you need to register your copy of Crystal Reports. You have two choices:

   Click **Next** to register now.

   Click **Register Later** to register later.

   After registering or choosing to register later, proceed to the next step.
2. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
3. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.

   The Standard Report Creation Wizard window appears.
4. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.
5. From the subfolder that opens, expand the **ODBC (RDO)** folder.
6. In the **ODBC (RDO)** dialog box, select the correct ODBC DSN entry for your version of Crystal Reports, as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Next**.
7. Leave the **User ID** and **Password** blank, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
8. Expand the **Tables** node and select the **Customer** table.
9. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
10. Expand the **Customer** table, then CTRL-click **Customer Name**, **Contact Title**, **Address1**, **Contact Last Name** and **City**.
11. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Next**.
12. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

The CustomersByCity report is created and loaded into the main window of Visual Studio.

You are now ready to add a parameter named City and populate it with default values.

To add a City parameter

The Field Explorer must be visible, because it provides access to the various features of the report, including parameters.

1. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click **Toggle Field View**.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.
2. In the **Field Explorer**, right-click **Parameter Fields** and select **New...**
3. In the **Create Parameter Field** dialog box:

   Set the **Name** to "City."

   Set the **Prompting Text** to "Select one or more cities."

   Set **Value Type** to **String**

   Select **Allow multiple values**.

   Select **Discrete value(s)**.

   Click **Default Values...**

   Note In Visual Studio .NET 2002 or 2003 where Crystal Reports has not been upgraded to Crystal Reports Developer, this button is named Set Default Values.
4. In the **Set Default Values** dialog box:

   Set the **Browse table** to "Customer."

   Set the **Browse field** to "City."

   Click **>>** (the double-right arrow) to move the entire cities list into the **Default Values** list.
5. Click **OK** to close the **Set Default Values** dialog box.
6. Click **OK** to close the **Create Parameter Field** dialog box.

You now use the Select Expert to set a formula that connects the City database column to your newly created City parameter field.

To connect the City parameter to the City database column

1. On the Crystal Reports toolbar, click **Select Expert**.
2. In the **Choose Field** dialog box, under **Report Fields**, select **Customer.City**, and then click **OK**.
3. In the **Select Expert** dialog box, within the **Customer.City** tab, set the list to "is equal to."
4. In the new list that appears to the right, select the first choice in the list, **{?City}**, and then click **OK**.

   Note This selection, {?City}, is the City parameter that you created earlier.
5. From the **File** menu, select **Save All**.

   At this point, it is advisable to view the report in full-screen mode.

6. To switch to full-screen mode, press Alt-Shift-Enter.
7. At the bottom of the window, click **Main Report Preview**.
8. In the **Enter Parameter Values** dialog box, select at least one city from the **Discrete Values** list, click **Add**, and then click **OK**.

   The report is displayed in preview mode, with the city parameter value that you selected.

   Note The Preview button is a new feature of the embedded Crystal Reports Designer that ships with Crystal Reports for Visual Studio 2005.
9. At the bottom of the window, click **Main Report** to exit Preview mode.
10. Press Alt-Shift-Enter to exit full-screen mode.
11. Open the Web Form.
12. From the **View** menu, click **Designer**.

    The report is displayed in Design Time Preview mode and shows the report's graphic layout on the Web Form.
13. From the **File** menu, click **Save All**.

Your report is created and is now the designated report for the CrystalReportSource control. You are now ready to build and run your project.

To test the reduced-code Web Site

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.
4. A parameters prompt page appears, asking you to select a prompt from a DropDownList.

   This is the expected result, for two reasons:

   You have not yet configured the Parameters tag in the report.

   In Smart Tasks for the CrystalReportViewer control, the "Enable report parameter prompting" checkbox is selected by default.
5. Select a parameter and click **Submit**.
6. The CustomersByCity report is displayed in the Web browser, showing the city you have selected.
7. Return to Visual Studio and click **Stop** to exit from debug mode.

Configuring Parameter Prompting in Smart Tasks

In this procedure, you examine the effect of turning off "Enable Report Parameter Prompting" in Smart Tasks.

To configure Report Parameter Prompting in Smart Tasks

1. Open the Default.aspx page (the Web Form).
2. From the **View** menu, click **Designer**.
3. On the upper-right corner of the CrystalReportViewer control, click on the small triangular button.

   The Smart Task panel named "CrystalReportViewer Tasks" opens.
4. Clear the **Enable Report Parameter Prompting** checkbox.
5. Click on the Web Form to close Smart Tasks.
6. From the **Build** menu, select **Build Solution**.
7. If you have any build errors, go ahead and fix them now.
8. From the **Debug** menu, click **Start**.

   Because report parameter prompting is no longer enabled, a "Missing Parameter Values" exception is thrown in the Web application window.
9. Return to Visual Studio and click **Stop** to exit from debug mode.
10. Return to the Smart Task panel for the CrystalReportViewer control.
11. Select the **Enable Report Parameters Prompting** checkbox.
12. Close Smart Tasks.
13. From the **File** menu, click **Save All**.

Preparing a Control with Parameter Values

In this section you prepare a DropDownList control with city values. In the next section, the city values are passed to the report parameters.

To prepare a control with parameter values

1. Open the Default.aspx page (the Web Form).
2. From the **View** menu, click **Designer**.
3. Click once on the **CrystalReportViewer** control to select it.
4. Press the LEFT ARROW.

   The flashing cursor appears at the lower-left of the **CrystalReportViewer** control.
5. Press Enter twice, then press the UP ARROW twice.

   The flashing cursor moves to the top of the Web Form, above the **CrystalReportViewer** control.
6. Type "Select a city:"
7. From the **Toolbox**, drag a **DropDownList** control to the right of the text.

   The Smart Task panel named "Common DropDownList Tasks" opens.
8. In the Smart Task panel, click **Edit Items**.
9. In the **List Item Collection Editor** dialog box, click **Add**.

   In the Members panel, a new ListItem entry is created.
10. In the **List Item Properties** panel, click into the **Text** field, and type "Paris."
11. Click **Add** again.

    In the Members panel, a second ListItem entry is created.

12. In the **List Item Properties** panel, click into the **Text** field, and type "Tokyo."
13. Click **OK**.
14. Click on the Web Form to close Smart Tasks for the **DropDownList** control.
15. Click **DropDownList** to select it.
16. From the **Properties** window, set the **ID** field to "cityList."
17. From the **Toolbox**, drag a **Button** control to the right of the **DropDownList** control.
18. From the **Properties** window,
    Set the **Text** field to "Redisplay Report."
    Set the **ID** field to "redisplay".
19. From the **File** menu, click **Save All**.

You now have a DropDownList control, with two city parameter values and a button to post values from this control. However, the report is not yet aware of this control and the values that it contains. In the next section, you connect the DropDownList control to the <Parameters> tag of the report.

Configuring Parameters for the CrystalReportSource Control

The CrystalReportSource control that you created in this tutorial implements the new Parameters model that is included with ASP.NET 2.0. Any parameter type that is part of the Data Parameters model can be passed as a parameter to the CrystalReportSource, including the following parameters:

ControlParameter
QueryStringParameter
SessionParameter
FormParameter
CookieParameter

In this tutorial, a ControlParameter is used. The DropDownList control that you created and populated with a list of cities in the previous section is configured as the parameter source for the report parameter contained in the CrystalReportSource control.

To configure the parameters for the CrystalReportSource control

1. Open the Default.aspx page (the Web form).
2. From the **View** menu, click **Designer**.
3. Select the **CrystalReportSource** control.
4. From **Properties**, expand **Report**.
5. Within the **Report** property, click the ellipsis (**...**) on the far right of **Parameters (Collection)**.
6. In the **Parameters Collection Editor** dialog box, do the following:
   a) Click **Add**.
   b) In the **Report** panel, select **Main report**.
   c) In the **Parameter Name** combo box, click **City**.
   d) Select **Specify Control ID**, and in the combo box click **cityList**.

7. Click **OK**.
8. At the bottom of the Default.aspx page, click **Source**.
9. Locate the CrystalReportSource control tagging information.
10. Within the CrystalReportSource tags, locate the Report tag.
11. Within the Report tags (between the opening and closing tags), verify the following Parameters tagging information.

```
<Parameters>
<CR:controlparameter name="City" propertyname="SelectedValue"
controlid="cityList"></CR:controlparameter>
</Parameters>
```

12. From the **Build** menu select **Build Solution**.
13. If you have any build errors, go ahead and fix them now.
14. From the **Debug** menu, click **Start**.

    The report is now successfully displayed with the parameter value from the default (first) selection in your list.
15. Change the parameter value to another value and click the **Redisplay Report** button.

    The report shows the city value you have selected in the DropDownList control.

    Note If you want to populate the DropDownList control with the default values for the City parameter from the CustomersByCity report, see the tutorial Reading and Setting Discrete Parameters.

Conclusion

In this reduced-code Web Site tutorial, you created and displayed a report that contained a City parameter. You enabled and disabled a report parameter prompt by configuring Smart Tasks. You also added a DropDownList control to the Web Form and added city values to the control. Then, you connected the DropDownList control to the report through the assignment of the control ID to the Parameters tag within the Report tag.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_ReducedCode_Parameters

Visual Basic Web Site: VB_Web_ReducedCode_Parameters

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorial:

Exposing Report Data to Other Controls in a Web Site

Reduced-Code Exposing Report Data to Other Controls in a Web Site

Introduction

In this reduced-code Web Site tutorial, you learn how to expose report data from the CrystalReportViewer control to other controls in the Web Site.

In this tutorial, you learn to expose Crystal report data from the CrystalReportViewer control to a label control in the Web Site.

To begin, you create an ASP.NET Web Site in Visual Studio and you add a CrystalReportViewer control to the Web Form. Then you add a label to the Web Form to display report data. From Smart Tasks, you create a CrystalReportSource control, and then add a sample report to it. You add a drill-down event to the CrystalReportViewer control. Finally, you expose the drill-down event's report data to the label control.

Creating a Web Site with a CrystalReportViewer control

Before you create a Web Site, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a reduced-code Web Site in Crystal Reports for Visual Studio 2005

1. Start Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Web Site**.
3. In the **New Web Site** dialog box, click **ASP.NET Web Site**.
4. In the **Location** dropdown, select **File System**.
5. In the **Language** dropdown, select the coding language that you wish to use.
6. In the **Location** text field enter the directory path "C:\WebSites\", followed by the name of your project.

   ```
   C:\WebSites\MyProjectName
   ```
7. Click **OK**.

   Your project is displayed in the Solution Explorer and contains a Default.aspx page.
8. Open the Default.aspx page (the Web Form).
9. From the **View** menu, click **Designer**.

   Note You can also switch to design view by clicking the Design button at the bottom of the form view.
10. Click on the Web Form and press **Enter** on your keyboard three times to create space for the **Label** control and for the **CrystalReportViewer** control.
11. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
12. Drag the **CrystalReportViewer** control onto the Web Form, to the bottommost space.

    The "CrystalReportViewer Tasks" Smart Task panel opens.
13. Press **Esc** on your keyboard to close the Smart Task panel.

14. Click the **CrystalReportViewer** control to select it.
15. From the **Properties** window, set the **ID** property:

    For Visual Basic Web Sites, set the **ID** property to "myCrystalReportViewer".

    For C# Web Sites, set the **ID** property to "crystalReportViewer".
16. From the **File** menu, click **Save All**.

Adding a Label Control to a Web Site

This section shows you how to add a label to the Web Form. The label that you add in this section shows the exposed report data later in this tutorial.

To add a label to the Web Site

1. If the Smart Task panel is currently open, press **Esc** on your keyboard to close it.
2. From the **Toolbox**, open the **Standard** node to locate the **Label** control.
3. Drag the **Label** control onto the Web Form, to the topmost space.
4. Click the **Label** control to select it.
5. From the **Properties** window, set the **ID** property to "drillLabel".
6. From the **File** menu, click **Save All**.

Configuring the CrystalReportSource in Smart Tasks

Visual Studio 2005 has a new GUI feature for .NET controls, called Smart Tasks. For the CrystalReportViewer control in Web Sites, the Smart Task panel is named "CrystalReportViewer Tasks."

The "CrystalReportViewer Tasks" Smart Task panel enables you to configure several features of the CrystalReportViewer control, without having to write code. Any selections that you make in Smart Tasks are generated as tag-based settings within the ASPX page.

In this section, you configure the main feature that is available from the "CrystalReportViewer Tasks" Smart Task: the CrystalReportSource control. Later in these reduced-code tutorials, you examine other features of Smart Tasks.

To begin, you learn how to close and re-open Smart Tasks.

To configure the CrystalReportSource using Smart Tasks

1. If the Smart Task panel is currently open, click on the Web Form to close the Smart Task panel.
2. On the upper-right corner of the **CrystalReportViewer** control, click on the small triangular button.

   The named "CrystalReportViewer Tasks" re-opens. In the **Choose Report Source** list, **<None>** is displayed.
3. Click the **Choose Report Source** list and select **<New report source...>**.

   The **Create ReportSource** dialog box opens.
4. In the **Specify a name for the control** textbox, leave the default entry as **CrystalReportSource1**.
5. Click the **Specify a report for the ReportSource control** drop- down list and select **<Browse...>**.

The **Select a Crystal Report** dialog box opens.

6. Navigate to the **World Sales Report.rpt** file in the General Business folder of the Crystal Reports sample reports directory. (See Appendix: Sample Reports' Directory for the locations of the sample reports.)

   Note The World Sales Report report retrieves its data from the Access database xtreme.mdb. If you have not verified the location of this database and its ODBC DSN configuration, see Appendix: What Needs to be Verified?.

7. Select the **World Sales Report.rpt** file and click **Open**.

   At this point, the report becomes the designated report for the **CrystalReportViewer** control.

8. From the **File** menu, click **Save All**.

Adding a Drill-Down Event to the CrystalReportViewer Control

This section shows you how to add a drill-down event to the CrystalReportViewer control so that when the user drills down on the report, the label can display the name of the report part that was drilled down on.

To add a drill-down event to the CrystalReportViewer control

1. From the **View** menu, click **Code** to view the code-behind class for this Web Form.
2. From the **View** menu, click **Designer** to go back to the designer view of the Web Form.
3. Click the **CrystalReportViewer** control to select it.
4. In the **Properties** window, click the **Events** button to view the list of events associated with the **CrystalReportViewer** control.
5. From the list of available events, under the **Action** node, locate the **Drill** event.
6. To generate the drill-down event handler code, double click on the **Drill** event in the **Properties** window.

   The following code is generated to handle the drill-down event:

[Visual Basic]

```
Protected Sub myCrystalReportViewer_Drill(ByVal source As Object, ByVal e
As CrystalDecisions.Web.DrillEventArgs) Handles
myCrystalReportViewer.Drill


End Sub
```

[end]

[C#]

```
protected void crystalReportViewer_Drill(object source,
CrystalDecisions.Web.DrillEventArgs e)
{


}
```

[end]

7. From the **File** menu, click **Save All**.

Exposing Drill-Down Event Report Data to the Label Control

This section shows you how to add text to the Label control to indicate which part of the report the user has drilled down on.

To expose drill-down event report data to the Label control

1. From the **View** menu, click **Code** to view the code-behind class for this Web Form.
2. Locate the following code that handles the drill-down event:

[Visual Basic]

```
Protected Sub myCrystalReportViewer_Drill(ByVal source As Object, ByVal e
As CrystalDecisions.Web.DrillEventArgs) Handles
myCrystalReportViewer.Drill


End Sub
```

[end]

[C#]

```
protected void crystalReportViewer_Drill(object source,
CrystalDecisions.Web.DrillEventArgs e)
{


}
```

[end]

3. To display the name of the drilled down report part on the **Label** control, add the following line of code in the event handler method:

[Visual Basic]

```
drillLabel.Text = e.NewGroupName
```

[end]

[C#]

```
drillLabel.Text = e.NewGroupName;
```

[end]

4. From the **File** menu, click **Save All**.

You are now ready to build and run the project.

To build and run the project

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, fix them now.
3. From the **Debug** menu, click **Start**.

   If no build errors appear, the project loads into your Web browser and displays the Web Form, with the World Sales Report generated on the form.

4. To test that the label displays the drill-down event report data correctly, under the **Top 5 Countries' Sales** heading, click on **USA** to drill down on this part of the report. The page reloads and the USA sales report displays, "USA".
5. Return to Visual Studio, and then click **Stop** to exit from debug mode.

Conclusion

In this reduced-code Web Site tutorial, you learned to add a CrystalReportViewer control to the Web Form, to use Smart Tasks to create a CrystalReportSource control, and to assign a Crystal report to the control. You added a drill-down event to the CrystalReportViewer control. Finally, you exposed the report data of that event to a label control in the event handler code.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_ReducedCode_ReportData

Visual Basic Web Site: VB_Web_ReducedCode_ReportData

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Reduced-Code Tutorial:

Exposing Report Data to Other Controls in a Windows Application

Reduced-Code Exposing Report Data to Other Controls in a Windows Application

Introduction

In this reduced-code tutorial, you learn how to expose report data from the CrystalReportViewer control to other controls in the Windows project.

In this tutorial, you learn to expose Crystal report data from the CrystalReportViewer control to a label control in the Windows project.

To begin, you learn how to create a Windows project in Crystal Reports for Visual Studio 2005. You then add a CrystalReportViewer control to the Windows Form. Then you add a label to the Windows Form to display report data. From Smart Tasks, you create a CrystalReportSource control, and then add a sample report to it. You add a drill-down event to the CrystalReportViewer control. Finally, you expose the drill-down event's report data to the label control.

Creating a Windows Project with a CrystalReportViewer Control

Before you create a Windows Project, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a reduced-code Windows project in Crystal Reports for Visual Studio 2005

1. Launch Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Project**.
3. In the **New Project** dialog box, select a language folder for C# or Visual Basic from the **Project Types** list.
4. From the **Templates** list, click **Windows Application**.
5. In the **Name** field, replace the default project name with the name of your project.
6. Click **OK**.

   Your project opens in Solution Explorer and contains a Form1 class.
7. Open the Form1 class.
8. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
9. Drag the **CrystalReportViewer** control onto the Windows Form.
10. From the **Properties** window, set the **Name** property:

    For Visual Basic projects, set the value to "myCrystalReportViewer."

    For C# projects, set the value to "crystalReportViewer."
11. From the **File** menu, click **Save All**.

Adding a Label Control to a Windows Project

This section shows you how to add a label to the Windows Form. The label that you add in this section shows the exposed report data later in this tutorial.

To add a label to the Windows application

1. If the Smart Task panel is currently open, press **Esc** on your keyboard to close it.
2. From the **Toolbox**, open the **Common Controls** node to locate the **Label** control.
3. Drag the **Label** control onto the Windows Form and place it directly above the **CrystalReportViewer** control.
4. Click the **Label** control to select it.
5. From the **Properties** window, set the **Name** property to "drillLabel".
6. From the **File** menu, click **Save All**.

Configuring the CrystalReportSource in Smart Tasks

Visual Studio 2005 has a new GUI feature for .NET controls, called Smart Tasks. For the CrystalReportViewer control in Web Sites, the Smart Task panel is named "CrystalReportViewer Tasks."

The "CrystalReportViewer Tasks" Smart Task panel enables you to configure several features of the CrystalReportViewer control, without having to write code. Any selections that you make in Smart Tasks are generated as tag-based settings within the ASPX page.

In this section, you configure the main feature that is available from the "CrystalReportViewer Tasks" Smart Task: the CrystalReportSource control. Later in these reduced-code tutorials, you examine other features of Smart Tasks.

To begin, you learn how to close and re-open Smart Tasks.

To configure the CrystalReportSource using Smart Tasks

1. If the Smart Task panel is currently open, click on the Windows Form to close the Smart Task panel.
2. On the upper-right corner of the **CrystalReportViewer** control, click on the small triangular button.
3. Click on the **Choose a Crystal Report...** option.

   The **Choose a Crystal Report** dialog box opens.
4. Click the **Specify a report for the ReportSource control** drop- down list and select **<Browse...>**.

   The **Select a Crystal Report** dialog box opens.
5. Navigate to the **World Sales Report.rpt** file in the General Business folder of the Crystal Reports sample reports directory. (See Appendix: Sample Reports' Directory for the locations of the sample reports.)

   Note The World Sales Report report retrieves its data from the Access database xtreme.mdb. If you have not verified the location of this database and its ODBC DSN configuration, see Appendix: What Needs to be Verified?.
6. Select the **World Sales Report.rpt** file and click **Open**.
7. Click **OK** to close the **Choose a Crystal Report** dialog box.

   At this point, the report becomes the designated report for the **CrystalReportViewer** control.
8. From the **File** menu, click **Save All**.

Adding a Drill-Down Event to the CrystalReportViewer Control

This section shows you how to add a drill-down event to the **CrystalReportViewer** control so that when the user drills down on the report, the label can display the name of the report part that was drilled down on.

To add a drill-down event to the CrystalReportViewer control

1. From the **View** menu, click **Code** to view the code-behind class for this Web Form.
2. From the **View** menu, click **Designer** to go back to the designer view of the Web Form.
3. Click the **CrystalReportViewer** control to select it.
4. In the **Properties** window, click the **Events** button to view the list of events associated with the **CrystalReportViewer** control.
5. From the list of available events, under the **Action** node, locate the **Drill** event.
6. To generate the drill-down event handler code, double click on the **Drill** event in the **Properties** window.

   The following code is generated to handle the drill-down event:

[Visual Basic]

```
Private Sub myCrystalReportViewer_Drill(ByVal source As System.Object,
ByVal e As CrystalDecisions.Windows.Forms.DrillEventArgs) Handles
myCrystalReportViewer.Drill


End Sub
```

[end]

[C#]

```
private void crystalReportViewer_Drill(object source,
CrystalDecisions.Windows.Forms.DrillEventArgs e)
{


}
```

[end]

5. From the **File** menu, click **Save All**.

Exposing Drill-Down Event Report Data to the Label Control

This section shows you how to add text to the **Label** control to indicate which part of the report the user has drilled down on.

To expose drill-down event report data to the Label control

1. From the **View** menu, click **Code** to view the code-behind class for this Windows Form.
2. Locate the following code that handles the drill-down event:

[Visual Basic]

```
Private Sub myCrystalReportViewer_Drill(ByVal source As System.Object,
ByVal e As CrystalDecisions.Windows.Forms.DrillEventArgs) Handles
myCrystalReportViewer.Drill


End Sub
```

[end]

[C#]

```
private void crystalReportViewer_Drill(object source,
CrystalDecisions.Windows.Forms.DrillEventArgs e)
{


}
```

[end]

6. To display the name of the drilled down report part on the **Label** control, add the following line of code in the event handler method:

[Visual Basic]

```
drillLabel.Text = "You drilled down on: " + e.NewGroupName
```

[end]

[C#]

```
drillLabel.Text = "You drilled down on: " + e.NewGroupName;
```

[end]

7. From the **File** menu, click **Save All**.

You are now ready to build and run the project.

To build and run the project

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, fix them now.
3. From the **Debug** menu, click **Start**.

   If no build errors appear, the project loads into your Windows application and displays the Windows Form, with the World Sales Report generated on the form.
4. To test that the label displays the drill-down event report data correctly, under the **Top 5 Countries' Sales** heading, double-click on **USA** to drill down on this part of the report. The page reloads and the label above the USA sales report displays, "USA".
5. Return to Visual Studio, and then click **Stop** to exit from debug mode.

Conclusion

In this reduced-code tutorial, you learned to add a CrystalReportViewer control to a Windows Form, to use Smart Tasks to create a CrystalReportSource control, and to assign a Crystal report to the control. You added a drill-down event to the CrystalReportViewer control. Finally, you exposed the report data of that event to a label control in the event handler code.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Windows Application: CS_Win_ReducedCode_ReportData

Visual Basic Windows Application: VB_Win_ReducedCode_ReportData

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Data Connectivity Tutorials

Crystal Reports

For Visual Studio 2005

Data Connectivity Tutorial:

Connecting to ADO.NET DataSets

Connecting to ADO.NET DataSets

Introduction

In this tutorial, you learn how to create a report that is connected to an ADO.NET DataSet and write code to display it in your application.

In this tutorial, you learn how to connect a Crystal report to an ADO.NET DataSet, through a DataSet schema. Some extra steps are required to report from an ADO.NET DataSet, because the report is not connected directly to a database.

An ADO.NET DataSet schema provides a template of the data structure in XML. However, a report cannot retrieve data from the DataSet schema alone. The DataSet schema must first be instantiated as a strongly-typed DataSet instance or as a generic DataSet instance. Then, the DataSet instance must be filled with data, through use of the DataAdapter classes.

In this tutorial, because of the added complexity that is involved in reporting from ADO.NET DataSets, the processes that you follow to create the schema and populate the DataSet are carefully kept separate from the Crystal Reports binding code.

> **Note** Crystal Reports for Visual Studio provides a template, named DataSet, to create an ADO.NET DataSet schema. In Visual Studio 2005 Web sites, the DataSet template is not accessible from the Add New Item dialog box. Only the default "XML Schema" template is available. However, with minor adjustments in the code (explained in the steps that follow), you can use the default template XML Schema to achieve the same results.

To begin, you create a data connection and build a DataSet schema that is based on that connection. You then create a helper class with a property that returns a populated instance of the DataSet. Finally, you write code that binds your Crystal report to the DataSet.

For the Crystal report binding code, you write code to do the following:

Instantiate the report.

Set its SetDataSource property to the populated DataSet property from the helper class.

Bind the populated Crystal report to the CrystalReportViewer control.

Finally, if you are building a Web Site, you place your populated DataSet instance into an ASP.NET Cache object, to improve performance and scalability.

It is recommended that you keep the DataSet configuration process and the Crystal report binding process distinct, both to maintain clarity in the code when you bind the report, and to promote reusability of the DataSet across your Web or Windows project.

Configuring a Connection to the Sample Database in the Server Explorer

In this tutorial, you use a GUI approach to generate your DataSet schema: in the main window of Visual Studio 2005, you drag a database table from Server Explorer onto a DataSet window.

> **Note** This process also works in Visual Studio .NET 2002 or 2003.

Your first step is to configure a database connection in the Server Explorer to the sample database that is provided with Crystal Reports.

To add a connection to the Xtreme sample database in Server Explorer

1. From the **View** menu, click **Server Explorer**.
2. In **Server Explorer**, right-click **Data Connections**, and then click **Add Connection...**

   The next steps are different depending on whether you are adding this connection in Visual Studio .NET 2002 or 2003, or in Visual Studio 2005.
3. In Visual Studio .NET 2002 or 2003, do the following:

   In the **Data Link Properties** dialog box, click the **Provider** tab.

   In the **Choose a data provider** panel, click **.NET Framework Data Provider for OLE DB**, and then click the **Connection** tab.

   In the **Select an OleDB provider** box, select **Microsoft Jet 4.0 OLE DB Provider**.
4. In Visual Studio 2005, do the following:

   If this is the first time you have added a connection in Visual Studio 2005 the **Change Data Source** window will appear. If the Add Connection dialog box appears, click **Change...**

   In the **Change Data Source** dialog box, in the **Data Provider** combo box, select **.NET Framework Data Provider for OLEDB**.

   In the **DataSource** list, select **Microsoft Access Database File**, and then click **OK**.
5. In the **Database File Name** field, enter the path to the Xtreme sample database.

   Note You can copy and paste the path to the Xtreme sample database. To access the path for your version of Crystal Reports, see Appendix: Location of Xtreme Sample Database.
6. Set the **User name** and **Password** fields to blank, and then click **Test Connection**.
7. If your test connection is not successful, recheck the path to the Xtreme sample database and ensure that both **User name** and **Password** are blank.
8. Click **OK** to close the **Test Connection** dialog box, and click **OK** again to close the **Connection Properties** dialog box.
9. In **Server Explorer**, expand the **Access** node, and then the **Tables** node.
10. Verify that the **Customer** table is visible in the **Server Explorer**.

Creating a DataSet Schema

You are now ready to create a new Windows project or Web Site, to which you will add an ADO.NET DataSet schema based on a table in the Access database.

To add an ADO.NET DataSet schema to a Windows project or Web Site

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In the **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** list, do the following:

   For a Windows project, select **DataSet**.

   For a Web Site, select **XML Schema**.
3. In the **Name** field, enter "CustomerDataSetSchema.xsd," and then click **Add**.
4. From the **Server Explorer**, drag the **Customer** table (located under the node **ACCESS>Tables>Customer**) onto the **CustomerDataSetSchema.xsd** window.
5. From the **Build** menu, click **Build Solution**.

   A strongly-typed DataSet class is generated from the schema.

   Note In a Visual Studio 2005 Web Site there is no strongly-typed DataSet class. Later in this tutorial you learn how to write code to generate your own DataSet class that is based on the CustomerDataSetSchema.xsd.
6. From the **File** menu, click **Save All**.

Writing a Helper Class to Populate the DataSet

The DataSet schema that you have just created for the Customer table is a data structure. At runtime, code is required to populate the DataSet structure with data from the database. In this section, you create a helper class that populates the DataSet with data.

To create a helper class to populate the DataSet with data

1. In the **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Visual Studio installed templates** list, select **Class**.
3. In the **Name** field, enter "DataSetConfiguration," and then click **Add**.
4. If a dialog box comes up and asks you whether to place your class in a directory named "Code," click **Yes**.
5. Above the class signature, add an `"Imports"[Visual Basic]` or `"using"[C#]` declaration to the top of the class for the System.Data and System.Data.OleDb namespaces.

[Visual Basic]

```
Imports System.Data
Imports System.Data.OleDb
```

[end]

[C#]

```
using System.Data;
using System.Data.OleDb;
```

[end]

6. At the top of the class, create a constant named CONNECTION_STRING to hold the connection string to the Xtreme sample database.

Note If you intend to copy and paste the code, note that the Data Source string is for Visual Studio 2005. To be sure that you have the correct file directory path to the xtreme.mdb database, see Appendix: Location of Xtreme Sample Database.

[Visual Basic]

```
Private Const CONNECTION_STRING As String =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Program Files\Microsoft
Visual Studio 8\Crystal Reports\Samples\En\Database\xtreme.mdb"
```

[end]

[C#]

```
private const string CONNECTION_STRING =
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Program
Files\\Microsoft Visual Studio 8\\Crystal
Reports\\Samples\\Database\\xtreme.mdb";
```

[end]

7. Beneath the first constant, create a second constant named QUERY_STRING to hold the database query string.

[Visual Basic]

```
Private Const QUERY_STRING As String = "SELECT * FROM CUSTOMER"
```

[end]

[C#]

```
private const string QUERY_STRING = "SELECT * FROM CUSTOMER";
```

[end]

Note This simple query string selects all columns, and no criteria. For the Xtreme sample database, this will return only a small amount of data. However, in most cases it is recommended to limit your query by including a WHERE clause and selecting a limited number of columns.

8. Beneath the second constant, create a third constant named DATATABLE_NAME for the name of the DataTable to be filled in the DataSet.

[Visual Basic]

```
Private Const DATATABLE_NAME As String = "Customer"
```

[end]

[C#]

```
private const string DATATABLE_NAME = "Customer";
```

[end]

The following step instructs you to create a DIRECTORY_FILE_PATH constant. This constant is only required if you are creating a Web Site in Visual Studio 2005. Otherwise, skip this step.

9. Beneath the third constant, create a fourth constant named DIRECTORY_FILE_PATH for referring to the directory path location of the xsd file.

   Note The code below demonstrates a path for a Web Site.

[Visual Basic]

```
    Private Const DIRECTORY_FILE_PATH As String =
    "C:\WebSites\VB_Web_Data_DataSets\"
```

[end]

[C#]

```
private const string DIRECTORY_FILE_PATH =
@"C:\WebSites\CS_Web_Data_DataSets\";
```

[end]

The next step procedure explains how to return a populated DataSet from the CustomerDataSet read-only property.

To create a property that populates the DataSet

1. Create a read-only property named CustomerDataSet that returns an instance of DataSet. Give the method a "Shared"[Visual Basic] or "static"[C#] modifier so that the class and property can be called directly without needing to be instantiated.

[Visual Basic]

```
Public Shared ReadOnly Property CustomerDataSet() As DataSet
    Get


    End Get
End Property
```

[end]

[C#]

```
public static DataSet CustomerDataSet
{
    get
    {


    }
}
```

[end]

2. This step has two options: one for use with the strongly-typed DataSet class (that is available with Visual Studio .NET 2002 or 2003 projects and with Visual Studio 2005 Windows projects.) and one for use with the generic DataSet class (that is available with Visual Studio 2005 Web Sites).

   For projects that use the strongly-typed DataSet class, within the get clause of the CustomerDataSet property, declare and instantiate the CustomerDataSetSchema class (the strongly-typed DataSet class that was generated from the DataSet schema in the previous section).

[Visual Basic]

```
    Dim myDataSet As CustomerDataSetSchema = New CustomerDataSetSchema()
```

[end]

[C#]

```
CustomerDataSetSchema dataSet = new CustomerDataSetSchema();
```

[end]

For projects that use the generic DataSet class in Visual Studio 2005 Web Sites, within the get clause of the CustomerDataSet property, declare and instantiate a DataSet class, and then apply the XML Schema to the DataSet instance. That is, pass the directory file path to the CustomerDataSetSchema.xsd as a string parameter to the `ReadXmlSchema()` method of the DataSet instance.

[Visual Basic]

```
Dim myDataSet As DataSet = New DataSet()
myDataSet.ReadXmlSchema(DIRECTORY_FILE_PATH & "XMLSchema.xsd")
```

[end]

[C#]

```
DataSet dataSet = new DataSet();
dataSet.ReadXmlSchema(DIRECTORY_FILE_PATH + "XMLSchema.xsd");
```

[end]

3. Declare and instantiate the OleDbConnection class and pass into it the CONNECTION_STRING constant as a method parameter.

[Visual Basic]

```
Dim myOleDbConnection As OleDbConnection = New
OleDbConnection(CONNECTION_STRING)
```

[end]

[C#]

```
OleDbConnection oleDbConnection = new OleDbConnection(CONNECTION_STRING);
```

[end]

4. Declare and instantiate the OleDbDataAdapter class and pass into it the QUERY_STRING constant and the OleDbConnection instance as method parameters.

[Visual Basic]

```
Dim myOleDbDataAdapter As OleDbDataAdapter = New
OleDbDataAdapter(QUERY_STRING, myOleDbConnection)
```

[end]

[C#]

```
OleDbDataAdapter oleDbDataAdapter = new OleDbDataAdapter(QUERY_STRING,
oleDbConnection);
```

[end]

5. Call the `Fill()` method of the OleDbDataAdapter instance and pass to it the CustomerDataSetSchema instance and the DATATABLE_NAME constant.

[Visual Basic]

```
myOleDbDataAdapter.Fill(myDataSet, DATATABLE_NAME)
```

[end]

[C#]

```
oleDbDataAdapter.Fill(dataSet, DATATABLE_NAME);
```

[end]

Note The Fill() method populates the specified DataTable, within the DataSet instance, with the data retrieved from the database.

6. To complete the property, return the DataSet instance.

[Visual Basic]

```
Return myDataSet
```

[end]

[C#]

```
return dataSet;
```

[end]

The CustomerDataSet property is created and can be called from anywhere in the project.

Creating a Report that Connects to the DataSet Schema

In the previous steps, you created a method that populates the DataSet schema with data from the database. In this section, you create a Crystal report that bases its data connectivity on the CustomerDataSetSchema schema.

To create a Crystal report that connects to the CustomerDataSetSchema schema

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Visual Studio installed templates** list, select **Crystal Report**.
3. In the **Name** field, enter the name "Customer.rpt" and click **Add**.

   Note In Visual Studio .NET 2002 or 2003, the button is named Open.
4. If you have not registered before, you are asked to register. To learn how to register, see Appendix: Crystal Reports Registration and Keycode.
5. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
6. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
7. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Project Data** folder.
8. Expand the **ADO.NET DataSets** node.
9. This step has two options: one for use with the strongly-typed DataSet class (that is available with Visual Studio .NET 2002 or 2003 projects and with Visual Studio 2005 Windows projects.) and one for use with the generic DataSet class (that is available with Visual Studio 2005 Web Sites).

   For projects that use the strongly-typed DataSet class, expand the **CustomerDataSetSchema** node. Expand the **Tables** node if there is one.

   For projects that use the generic DataSet class in Visual Studio 2005 Web Sites, expand the **DataSetConfiguration** node.

In the **ADO.NET (XML)** dialog box, select **Use DataSet from Class**. In the DataSet Names dialog box, select the **CustomerDataSet** property, and then click **Finish**.

10. Select the **Customer** table node.
11. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
12. Expand the **Customer** table, then CTRL-click **Customer Name**, **Contact Title**, **Address1**, **Customer Las Name** and **City**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click the **Next** button.
14. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click the **Finish** button.

    The Customer report is created, with data connectivity to the CustomerDataSetSchema (or, in a Visual Studio 2005 Web Site, to the CustomerDataSet property of the DataSetConfiguration class.)

Binding the Report and Setting the DataSource to the Populated DataSet

In this section, you learn how to instantiate the report that you have created, populate the report's DataSet, and display the report in the CrystalReportViewer control. You populate the report through the assignment of its SetDataSource property to the populated DataSet, which is returned from the CustomerDataSet property of the DataSetConfiguration class. Finally, you bind the populated report to the CrystalReportViewer control.

You can instantiate and bind the report in the following ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

To instantiate and bind an embedded report to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Above the class signature, add an `"Imports"[Visual Basic]` or `"using"[C#]` declaration to the top of the class for the System.Data namespace if it is not there already.

[Visual Basic]

```
Imports System.Data
```

[end]

[C#]

```
using System.Data;
```

[end]

4. Add a new class-level declaration for the Customer report wrapper class, with the variable name customerReport. Set its access modifier to private.

[Visual Basic]

```
Private customerReport As Customer
```

[end]

[C#]

```
private Customer customerReport;
```

[end]

5. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
customerReport = New Customer()
```

[end]

[C#]

```
customerReport = new Customer();
```

[end]

6. On the next line beneath the report instantiation, declare a DataSet.

[Visual Basic]

```
Dim myDataSet As DataSet
```

[end]

[C#]

```
DataSet dataSet;
```

[end]

   This step and the next step separate the declaration of the variable from the assignment of the variable. Each line of code is kept separate because, in a Web Site addendum to this tutorial, you will refactor the variable assignment into a code block that caches the DataSet in the ASP.NET Cache object.

7. Assign the DataSet instance to the DataSetConfiguration.CustomerDataSet property.

[Visual Basic]

```
myDataSet = DataSetConfiguration.CustomerDataSet
```

[end]

[C#]

```
dataSet = DataSetConfiguration.CustomerDataSet;
```

[end]

8. Call the `SetDataSource()` method of the CustomerReport instance and pass into it the DataSet instance.

[Visual Basic]

```
customerReport.SetDataSource(myDataSet)
```

[end]

[C#]

```
customerReport.SetDataSource(dataSet);
```

[end]

9. Bind the ReportSource property of the CrystalReportViewer control to the CustomerReport instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customerReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customerReport;
```

[end]

You are now ready to build and run your project. Skip over the non-embedded report step procedure to the step procedure that follows it.

To instantiate and bind a non-embedded report to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, with the variable name customerReport. Set its access modifier to private.

[Visual Basic]

```
Private customerReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument customerReport;
```

[end]

> **Note** The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You added an `"Imports"[Visual Basic]` or `"using"[C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (which you added during one of the procedures in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
customerReport = New ReportDocument()
```

[end]

[C#]

```
customerReport = new ReportDocument();
```

[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("Customer.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("Customer.rpt");
```

[end]

For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"Customer.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\" + "Customer.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
customerReport.Load(reportPath)
```

[end]

[C#]

```
customerReport.Load(reportPath);
```

[end]

7. Declare a DataSet and assign to it the DataSetConfiguration.CustomerDataSet property.

[Visual Basic]

```
Dim myDataSet As DataSet = DataSetConfiguration.CustomerDataSet
```

[end]

[C#]

```
DataSet dataSet = DataSetConfiguration.CustomerDataSet;
```

[end]

8. Call the `SetDataSource()` method of the customerReport ReportDocument instance and pass into it the DataSet instance.

[Visual Basic]

```
customerReport.SetDataSource(myDataSet)
```

[end]

[C#]

```
customerReport.SetDataSource(dataSet);
```

[end]

9. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customerReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customerReport;
```

[end]

You are now ready to build and run your project.

To test the loading of the Customer report and its populated DataSet

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

   Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.
4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.
5. The Customer report displays and shows the populated data that you placed in the DataSet.
6. Return to Visual Studio and click **Stop** to exit from debug mode.

If you are building a Windows project, continue to Conclusion.

If you are building a Web Site, continue to Caching the DataSet in a Web Site.

Caching the DataSet in a Web Site

If you are building a Web Site, you can gain additional scalability and performance by placing your populated DataSet instance into the ASP.NET Cache object. This prevents redundant (and potentially slow) calls to the database to populate the DataSet.

Note If your DataSet contains standard values for all users, use the Cache object. However, if your DataSet contains unique values for each user (based on a user-specific criteria in the WHERE clause of the SQL query, such as user id) then you should use the Session object instead.

In this section, you expand one line of code that assigns the CustomerDataSet property to the DataSet instance to a complete code block that manages DataSet caching.

To modify your Web Site to check for a cached DataSet

1. Open the Web Form.
2. From the **View** menu, click **Code**.
3. Locate the line of code where the DataSet instance is assigned to the DataSetConfiguration.CustomerDataSet property (shown below).

[Visual Basic]

```
    myDataSet = DataSetConfiguration.CustomerDataSet
```

[end]

[C#]

```
    dataSet = DataSetConfiguration.CustomerDataSet;
```

[end]

4. Replace this line of code with a complete conditional code block that checks for a Cache value named "customerDataSet."

   Enter the conditional block and code exactly as shown here.

[Visual Basic]

```
    If Cache("customerDataSet") Is Nothing Then
        myDataSet = DataSetConfiguration.CustomerDataSet
        Cache("customerDataSet") = myDataSet
    Else
        myDataSet = CType(Cache("customerDataSet"), DataSet)
    End If
```

[end]

[C#]

```
    if (Cache["customerDataSet"] == null)
    {
        dataSet = DataSetConfiguration.CustomerDataSet;
        Cache["customerDataSet"] = dataSet;
    }
    else
    {
        dataSet = (DataSet)Cache["customerDataSet"];
    }
```

[end]

You are now ready to build and run your Web Site to test the cached DataSet.

To test the caching of the DataSet

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The Customer report displays and shows the populated data that you placed in the DataSet.
4. Click your browser's **Refresh** button.

   The Customer report displays again quickly, because the DataSet is now retrieved from the ASP.NET Cache object.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

In this tutorial you learned how to connect a report to an ADO.NET DataSet schema and populate the DataSet schema with data and then display it in the report at runtime.

You added a connection to the Xtreme sample database in the Server Explorer. You dragged a Customer table onto a DataSet window to create a CustomerDataSetSchema.xsd. Then you wrote a helper class that returned a populated DataSet at runtime.

You created a Crystal report and configured it to address the CustomerDataSetSchema schema (or, for a Visual Studio 2005 Web Site, a DataSet configured to the CustomerDataSetSchema.xsd file). You then wrote code in the ConfigureCrystalReports() method to retrieve a filled DataSet from the CustomerDataSet property in the helper class and pass that DataSet into the report before binding the report to the CrystalReportViewer control.

In an addendum for Web Sites, you placed the populated DataSet instance into the ASP.NET Cache object, to improve performance and scalability.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_Data_DataSets

C# Windows project: CS_Win_Data_DataSets

Visual Basic Web Site: VB_Web_Data_DataSets

Visual Basic Windows project: VB_Win_Data_DataSets

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Data Connectivity Tutorial:

Connecting to IDataReader

Connecting to IDataReader

Introduction

In this data connectivity tutorial, you learn how to connect to an instance of IDataReader.

Many .NET projects use DataReader, rather than DataSet, to retrieve data. For example, a typical way to bind a control (such as the GridView control) to data is to create a method that retrieves data through a DataReader, and then return that data from the method through an IDataReader interface.

In Crystal Reports for Visual Studio 2005, the embedded Crystal Report Designer can access IDataReader, provided certain conditions are met:

The data must be returned from a method, through the IDataReader interface.

The data provider that is used within the method must be the OleDb .NET data provider.

Note The classes for this data provider are located in the System.Data.OleDb namespace.

The method must be Shared [Visual Basic] or static [C#].

The class that contains this method must be part of a class library project and be compiled into an assembly.

In this tutorial, you learn how to connect a Crystal report to the DLL assembly file, and how to access a class and static method within the assembly that returns the IDataReader.

To begin, you create a class library, add a class and a static method that returns IDataReader, and then compile that class library into an assembly.

Creating the Class Library

Some setup is required as a prerequisite to this tutorial.

Prerequisite Database Setup

1. SQL Server configuration:

   If you have SQL Server (or the OEM version, MSDE) installed, it must be configured to require SQL Server Authentication.

   If you do not have SQL Server (or the OEM version, MSDE) installed, you must install MSDE with SQL Server Authentication set to "True."
2. The Northwind database provided with SQL Server must be installed and verified that it requires SQL Server Authentication.
3. A limited access account must be created for use within the Web site.

To install MSDE with SQL Server Authentication, or the Northwind database, go to the following sections from Appendix: System Setup in this documentation:

Appendix: MSDE Installation with Windows or SQL Server Authentication

Appendix: Northwind Database Installation

Appendix: Security: Creating a Limited Access Database Account

Once you have configured SQL Server and the Northwind database, you are ready to create a class library that returns values from the Northwind database as IDataReader.

Note This tutorial uses the Northwind database with SQL Server, but you can use any OleDb-compliant database.

To create a class library containing a static method that returns IDataReader

1. From the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project** dialog box, select a language folder for C# or Visual Basic from the **Project Types** list.
3. From the **Templates** list, click **Class Library**.
4. In the **Name** field, replace the default project name with `"VB_Lib_DataLayer" [Visual Basic]` or `"CS_Lib_DataLayer" [C#]`, and then click **OK**.
5. In **Solution Explorer**, right-click the default class (Class1), and then click **Delete**.
6. In **Solution Explorer**, right-click the project name in bold type, point to **Add**, and then click **Class**.
7. In the **Add New Item** dialog box, in the **Templates** view, select the template named "Class."
8. In the Name field, type "DataCenter," and then click **Add**.
9. If you are using Visual Basic, type "Option Strict On" at the top of the class.
10. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class for the System.Data and System.Data.OleDb namespaces (if these namespace have not already been declared).

[Visual Basic]

```
Imports System.Data
Imports System.Data.OleDb
```

[end]

[C#]

```
using System.Data;
using System.Data.OleDb;
```

[end]

11. At the class level, create a private string constant for the OleDb connection string.

    Note For security reasons, it is important that you use a database account with limited access permissions. For more information, see Appendix: Security: Creating a Limited Access Database Account.

    In the code that you write, replace the sample server name *DBServer01* and the sample password *1234* (shown below) with the server name and password for your database.

[Visual Basic]

```
Private Const OLEDB_CONNECTION_STRING As String = "provider=sqloledb;Data
Source=DBSERVER01;Initial Catalog=Northwind;User
Id=limitedPermissionAccount;Password=1234"
```

[end]

[C#]

```
private const string OLEDB_CONNECTION_STRING = "provider=sqloledb;Data
Source=DBSERVER01;Initial Catalog=Northwind;User
Id=limitedPermissionAccount;Password=1234";
```

[end]

12. Create a second private string constant for the database query.

[Visual Basic]

```
Private Const CUSTOMER_TABLE_QUERY As String = "SELECT CustomerID,
CompanyName, ContactName, Address, City FROM Customers"
```

[end]

[C#]

```
private const string CUSTOMER_TABLE_QUERY = "SELECT CustomerID,
CompanyName, ContactName, Address, City FROM Customers";
```

[end]

13. Create a public `"Shared" [Visual Basic]` or `"static" [C#]` method named `GetCustomersUsingOleDb()` that returns IDataReader.

[Visual Basic]

```
Public Shared Function GetCustomersUsingOleDb() As IDataReader
End Function
```

[end]

[C#]

```
public static IDataReader GetCustomersUsingOleDb()
{
}
```

[end]

14. Within this method, declare and instantiate OleDbConnection and pass in the OLEDB_CONNECTION_STRING constant.

[Visual Basic]

```
Dim myOleDbConnection As OleDbConnection = New
OleDbConnection(OLEDB_CONNECTION_STRING)
```

[end]

[C#]

```
OleDbConnection oleDbConnection = new
OleDbConnection(OLEDB_CONNECTION_STRING);
```

[end]

15. Call the `Open()` method from the OleDbConnection instance to open the database connection.

[Visual Basic]

```
myOleDbConnection.Open()
```

[end]

[C#]

```
oleDbConnection.Open();
```

[end]

16. Declare and instantiate OleDbCommand and pass in the CUSTOMER_TABLE_QUERY constant and the OleDbConnection instance.

[Visual Basic]

```
Dim myOleDbCommand As OleDbCommand = New
OleDbCommand(CUSTOMER_TABLE_QUERY, myOleDbConnection)
```

[end]

[C#]

```
OleDbCommand oleDbCommand = new OleDbCommand(CUSTOMER_TABLE_QUERY,
oleDbConnection);
```

[end]

17. Call the `ExecuteReader()` method of the OleDbCommand instance and pass it to an instance of IDataReader.

[Visual Basic]

```
Dim myIDataReader As IDataReader = myOleDbCommand.ExecuteReader()
```

[end]

[C#]

```
IDataReader iDataReader = oleDbCommand.ExecuteReader();
```

[end]

18. Return the IDataReader instance from the method.

[Visual Basic]

```
Return myIDataReader
```

[end]

[C#]

```
return iDataReader;
```

[end]

19. From the **Build** menu, click **Build Solution**.
20. From the **File** menu, click **Close Solution**.

Connecting a Report to the IDataReader Static Method

In this section you create a client project that accesses the IDataReader within the compiled assembly.

To connect a report to the IDataReader static method

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **New Item**.
2. In the **Add New Item** dialog box, select **Crystal Report**.

3. In the **Name** field, enter "CustomersViaIDR.rpt", and then click **OK**.
4. In the **Crystal Reports Gallery** dialog box, click **OK**.
5. In the **Standard Report Creation Wizard** dialog box, expand **Create New Connection** node.
6. Expand the **ADO.NET** node.
7. In the **ADO.NET** dialog box, click the **...** at the end of the **File Path** text field.
8. In the **Open** dialog box, set the **Files of type** list to **All Files**.
9. Locate the DLL assembly from the following file directory path.

   Note The DLL assembly is created in the previous procedure, Creating the Class Library.

[Visual Basic]

```
\My Documents\Visual Studio
2005\Projects\VB_Lib_IDataReader\VB_Lib_IDataReader\bin\VB_Lib_DataLaye
r.dll
```

[end]

[C#]

```
\My Documents\Visual Studio
2005\Projects\CS_Lib_DataReader\CS_Lib_IDataReader\bin\Debug\CS_Lib_Dat
aLayer.dll
```

[end]

   Note In Visual Basic, the DLL is located in the bin directory. In C#, the DLL is located in the bin subdirectory named Debug.

10. Select the DLL, and then click **Open**.
11. From the **Class Name** list, select the **DataCenter** class, and then click **Finish**.

    In the Available Data Sources area, the GetCustomersUsingOleDB() method appears.
12. Select **GetCustomersUsingOleDb** and click the **>** button to move the method into the **Selected Tables** panel, and then click **Next**.

    If the IDataReader class library has not been correctly configured, an exception is thrown.

If your class library IDataReader method throws an exception, continue to Dealing With Exceptions.

Otherwise, continue to Completing the Report.

Dealing With Exceptions

In this section, you learn how to deal with exceptions that may be thrown by your class library IDataReader method.

To deal with exceptions

1. Close your Web Site or Windows project, and then reopen the class library project that contains the IDataReader static method.

   Note The class library project that is created in Creating the Class Library, is called `"VB_Lib_DataLayer" [Visual Basic]` or `"CS_Lib_DataLayer" [C#]`.

2. Check the connection string constant for errors in the database server name or password.
3. Correct any errors.
4. Check the query string constant for errors in the query.
5. Correct any errors.
6. From the **Build** menu, click **Build Solution**.
7. From the **File** menu, click **Close Solution**.
8. Reopen the Web Site or Windows project.
9. Delete the Crystal report file that you have created, and then create a new Crystal Report.
10. Repeat the steps in the previous section, Connecting a Report to the IDataReader Static Method.

The GetCustomersUsingOleDb() method is successfully transferred to the Selected Tables panel.

Note If you still get an exception, repeat the debugging process.

To debug further, create a new Web or Windows Form in your Web Site or Windows project and add a GridView control to that form. In the code-behind class, set the DataSource property of the GridView control to the IDataReader static method. Then, compile and view that Web or Windows Form to determine whether the display is successful.

Resolve any exceptions.

Completing the Report

You are now ready to design the report, based on values from the IDataReader static method.

To design the report based on values from the IDataReader static method

Note This step procedure assumes that you have successfully accessed the GetCustomersUsingOleDb IDataReader static method and, therefore, the method has been transferred into the Selected Tables panel of the Standard Report Creation Wizard dialog box.

1. In the **Available Fields** panel, expand **GetCustomersUsingOleDb**.
2. CTRL-click **CompanyName**, **ContactName** and **City**.
3. Click the **>** symbol to move these fields into the **Fields to Display** panel, and then click **Next**.
4. In the **Available Fields** panel, under **Report Fields**, select **GetCustomerUsingOleDb.City**, click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

   The CustomersViaIDR report is created and loaded into the main window of Visual Studio.
5. Click **Main Report Preview** to preview the report, based on a call to the **IDataReader** static method.

You are now ready to bind the report to the CrystalReportViewer control.

Binding the Report

In Appendix: Project Setup, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous step, you have added a CustomersViaIDR report to the project.

In this section, you instantiate the CustomersViaIDR report and bind it to the CrystalReportViewer control.

You can instantiate and bind the report in two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the CustomersViaIDR report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the CustomersViaIDR report as a non-embedded report.

To instantiate the CustomersViaIDR report as an embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the CustomersViaIDR report wrapper class, using the variable name customersViaIdrReport. Set its access modifier to private.

[Visual Basic]

```
Private customersViaIdrReport As CustomersViaIDR
```

[end]

[C#]

```
private CustomersViaIDR customersViaIdrReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
customersViaIdrReport = New CustomersViaIDR()
```

[end]

[C#]

```
customersViaIdrReport = new CustomersViaIDR();
```

[end]

5. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer control to the instantiated report class (variable name, customersViaIdrReport).

[Visual Basic]

```
    myCrystalReportViewer.ReportSource = customersViaIdrReport
```
[end]

[C#]

```
    crystalReportViewer.ReportSource = customersViaIdrReport;
```
[end]

You are now ready to build and run your project. Skip to the next section below.

To instantiate the CustomersViaIDR report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, using the variable name customersViaIdrReport. Set its access modifier to private.

[Visual Basic]

```
    Private customersViaIdrReport As ReportDocument
```
[end]

[C#]

```
    private ReportDocument customersViaIdrReport;
```
[end]

> **Note** The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
    customersViaIdrReport = New ReportDocument()
```
[end]

[C#]

```
    customersViaIdrReport = new ReportDocument();
```
[end]

5. Declare a string variable, name it reportPath, and assign to it a runtime path to the local report. This path is determined differently for Web Sites and Windows projects:

   For a Web Site, pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the hard drive file directory path at runtime.

[Visual Basic]

```
    Dim reportPath As String = Server.MapPath("CustomersViaIDR.rpt")
```
[end]

[C#]

```
    string reportPath = Server.MapPath("CustomersViaIDR.rpt");
```

[end]

For a Windows project, concatenate the `Application.StartupPath` property with a backslash and the local report file name. This maps the report to the same directory as the Windows executable file.

Note At compile time you will copy the report to the directory containing the executable file.

[Visual Basic]

```
Dim reportPath As String = Application.StartupPath & "\" &
"CustomersViaIDR.rpt"
```

[end]

[C#]

```
string reportPath = Application.StartupPath + "\\"
+"CustomersViaIDR.rpt";
```

[end]

6. Call the `Load()` method of the ReportDocument instance and pass into it the reportPath string variable.

[Visual Basic]

```
customersViaIdrReport.Load(reportPath)
```

[end]

[C#]

```
customersViaIdrReport.Load(reportPath);
```

[end]

7. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = customersViaIdrReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = customersViaIdrReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: customersViaIdrReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project.

To test the loading of the CustomersViaIDR report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. If you use a non-embedded report in a Windows project, locate the compiled Windows executable in the `\bin\ [Visual Basic]` or `\bin\debug\ [C#]` subdirectory, and then copy the report to that subdirectory.

Note To have the non-embedded report loaded by the Windows executable at runtime, the report must be stored in the same directory as the Windows executable.

4. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The report is displayed, showing data from the IDataReader static method.

5. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

In this data connectivity tutorial, you learned how to connect to IDataReader. You created a class library and populated it with a static method that returns IDataReader from an OleDb database connection. You compiled the class library to generate a DLL assembly file.

You then created a new report with the embedded Crystal Report Designer and connected to the IDataReader static method within the DLL.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_Data_IDataReader

C# Windows project: CS_Win_Data_IDataReader

Visual Basic Web Site: VB_Web_Data_IDataReader

Visual Basic Windows project: VB_Win_Data_IDataReader

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Data Connectivity Tutorial:

Connecting to Object Collections

Connecting to Object Collections

Introduction

In this tutorial on data connectivity, you learn how to use an object collection as the data source for a Crystal report.

In this tutorial, you create a class that is the type for each object in the object collection. The class will represent stock market information. When you build your Crystal report, this Stock class is accessed from the report wizard much like a database table, but rather than add table columns as fields to display, you add class properties instead.

When the report is first displayed, the report is empty. The report design is complete, but no data is available to populate the report.

Then you create a method that instantiates an ArrayList and adds multiple Stock instances to the ArrayList instance. Each Stock instance has its properties set to unique values. The ArrayList instance is then returned from the method. You will add this information programmatically at design time, and again dynamically at run time.

The returned ArrayList, an object collection, is assigned to the SetDataSource property of the Crystal report. When the report is displayed, each object in the object collection provides one Detail row in the report.

To begin, you create the Stock class.

Creating the Stock Class

In this section, you create the Stock class. The Stock class contains private fields that are exposed as public properties: Symbol, Volume, and Price.

To create the Stock class

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the web site name that is in bold type and then click **Add New Item**.
2. In the **Add New Item** dialog box:

   In the **Visual Studio Installed Templates** field, select **Class**

   In the **Name** field, type "Stock", and then click **Add**.

   In the dialog box that appears, click **Yes**.

   Note In Visual Studio 2005, all classes must be placed inside of an `App_Code` folder if they are to be generally consumable. When you click the **Add** button, an alert box will ask you if you would like to place your class inside of this `App_Code` folder.

   The Stock class must be set to public scope to be accessed when you create the report. Verify that the class you have created is public. If not, add the public modifier to the class signature to expose the class to the embedded Crystal Report Designer.

[Visual Basic]

```
Public Class Stock

End Class
```

[end]

[C#]

```
public class Stock
{
    public Stock()
    {
    }
}
```

[end]

3. If coding in Visual Basic, add a default constructor.

[Visual Basic]

```
    Sub New()
    End Sub
```

[end]

4. Within the class, add three private fields.

[Visual Basic]

```
Private _symbol As String
Private _volume As Integer
Private _price As Double
```

[end]

[C#]

```
private string _symbol;
private double _price;
private int _volume;
```

[end]

Next, you will add three public read/write properties to encapsulate the three private fields.

5. Create a new property named Symbol.

[Visual Basic]

```
Public Property Symbol() As String
    Get
        Return _symbol
    End Get
    Set(ByVal value As String)
        _symbol = value
    End Set
```

```
    End Property
```
[end]
[C#]
```
    public string Symbol
    {
        get
        {
              return _symbol;
        }
        set
        {
              _symbol = value;
        }
    }
```
[end]

6. Create a new property named Price.

[Visual Basic]
```
    Public Property Price() As Double
        Get
              Return _price
        End Get
        Set(ByVal value As Double)
              _price = value
        End Set
    End Property
```
[end]
[C#]
```
    public double Price
    {
        get
        {
              return _price;
        }
        set
        {
              _price = value;
```

```
        }
    }
```

[end]

7. Create a new property named Volume.

[Visual Basic]

```
    Public Property Volume() As Integer
        Get
            Return _volume
        End Get
        Set(ByVal value As Integer)
            _volume = value
        End Set
    End Property
```

[end]

[C#]

```
    public int Volume
    {
        get
        {
            return _volume;
        }
        set
        {
            _volume = value;
        }
    }
```

[end]

8. Finally, create a new constructor that takes the three public properties as arguments.

[Visual Basic]

```
    Sub New(ByVal symbol As String, ByVal volume As Integer, ByVal price As
    Double)
        _symbol = symbol
        _volume = volume
        _price = price
    End Sub
```

[end]

[C#]

```
    public Stock (String symbol, int volume, double price)
    {
        _symbol = symbol;
        _volume = volume;
        _price = price;
    }
[end]
```

9. From the **Build** menu, click **Build Website**.

   If you have any build errors, fix them now.

You are now ready to access this object from the embedded Crystal Report Designer.

Connecting a Report to the Stock Object

In this section, you create a new Crystal report in the embedded Crystal Report Designer and connect the report to the Stock object.

To connect a Crystal report to the Stock object

1. Right-click the project name and click **Add New Item**.
2. In the **Add New Item** dialog box, select **Crystal Report**.
3. In the **Name** field, enter "StockObjects.rpt", and then click **Add**.
4. In the **Crystal Reports Gallery** dialog box, click **OK**.
5. In the **Standard Report Creation Wizard** dialog box, expand **Project Data** and the sub node **.NET Objects**.

   A list of classes within the project appears. Each class is prefixed with the project namespace.
6. Expand the **Stock** class to view a selectable sub node.
7. Click the right arrow to move the Stock class sub node to the **Selected Tables** panel, and then click **Next**.
8. Expand **Stock** and click the **>>** to move all columns to the **Fields to Display** panel, and then click **Next**.
9. Select **Symbol** and click the right-arrow to move into the **Group By** panel, and then click **Finish**.

Binding the Report

In Appendix: Project Setup, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous procedure, you added a StockObjects report to the project.

In this section, you will bind the Stock Objects report to the Crystal Report Viewer, set the data source of the report to an Object Collection, and populate the Object Collection programmatically.

To instantiate the StockObjects report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the default Code-Behind class, `Default.aspx.cs` or `Default.aspx.vb`.
2. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class for the `System.Collections` namespace.

[Visual Basic]

```
Imports System.Collections
```

[end]

[C#]

```
using System.Collections;
```

[end]

> **Note** This reference gives you access to the `ArrayList` class. `ArrayList` implements `ICollection`. This qualifies `ArrayList` as one of several class types that can be used to build an object collection that is recognized by Crystal Reports.

3. Add a new class-level `ArrayList`, call it `stockValues`.

[Visual Basic]

```
Private stockValues As ArrayList
```

[end]

[C#]

```
private ArrayList stockValues;
```

[end]

4. Add a new class-level declaration for the ReportDocument report wrapper class, with the variable name `stockObjectsReport`. Set its access modifier to private.

[Visual Basic]

```
Private stockObjectsReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument StockObjectsReport;
```

[end]

5. Within the `ConfigureCrystalReports()` method you created in Appendix: Project Setup, declare a string variable, name it `reportPath`, and assign to it a runtime path to the local report.

   Pass the name of the local report file as a string parameter into the `Server.MapPath()` method. This maps the local report to the file path at runtime.

[Visual Basic]

```
Dim reportPath As String = Server.MapPath("StockObjects.rpt")
```

[end]

[C#]

```
string reportPath = Server.MapPath("StockObjects.rpt");
```

[end]

6. Add two line breaks, and instantiate the `ReportDocument` class.

[Visual Basic]

```
    stockObjectsReport = New ReportDocument()
```
[end]
[C#]
```
    stockObjectsReport = new ReportDocument();
```
[end]

7. On the next line, Call the `Load()` method of the `ReportDocument` instance and pass into it the `reportPath` string variable.

[Visual Basic]
```
    stockObjectsReport.Load(reportPath)
```
[end]
[C#]
```
    stockObjectsReport.Load(reportPath);
```
[end]

> **Note** The `ReportDocument` class is a member of the `CrystalDecisions.CrystalReports.Engine` namespace. You added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Project Setup. When you instantiate `ReportDocument` and load a report, you gain access to the report through the SDK.

8. Next, set the data source of the report to the `stockValues ArrayList`.

[Visual Basic]
```
    stockObjectsReport.SetDataSource(stockValues)
```
[end]
[C#]
```
    stockObjectsReport.SetDataSource(stockValues);
```
[end]

9. Finally, bind the `ReportSource` property of the `CrystalReportViewer` to the `ReportDocument` instance.

[Visual Basic]
```
    myCrystalReportViewer.ReportSource = stockObjectsReport
```
[end]
[C#]
```
    crystalReportViewer.ReportSource = stockObjectsReport;
```
[end]

At this point, the Stock Objects report is bound to the Crystal Report Viewer and the page displays the correct report; however, the report is currently bound to an empty data source, thus the report has no information to display. In the next section, you will programmatically populate the `stockValues ArrayList` with sample data.

Populating the Object Collection Programmatically

In this section, you will add Session code to the ASPX code-behind class. If there are no values currently held in session, default values will be created. If there are values in session, they will be assigned to the `stockValues ArrayList`.

1. Right click on the web form in the **Solution Explorer** and click **View Code**.
2. Within the class, add a new public scope helper method, with no return value, named `PopulateStockValuesArrayList()`.

[Visual Basic]

```
Public Sub PopulateStockValuesArrayList()


End Sub
```

[end]

[C#]

```
public void PopulateStockValuesArrayList()
{
}
```

[end]

3. Within the `PopulateStockValuesArrayList()` method, before the existing code, create an if/else conditional block that checks whether a Session object named `stockValues` exists.

[Visual Basic]

```
If (Session("stockValues") Is Nothing) Then


Else


End If
```

[end]

[C#]

```
if(Session["stockValues"] == null)
{
}
else
{
}
```

[end]

4. Within the If block, instantiate a new `ArrayList()`

[Visual Basic]

```
stockValues = New ArrayList
```

[end]

[C#]

```
stockValues = new ArrayList();
```

[end]

5. Next, use the overloaded constructor for the Stock class to create and instantiate three instances of Stock.

[Visual Basic]

```
Dim s1 As Stock = New Stock("AWRK", 1200, 28.47)
Dim s2 As Stock = New Stock("CTSO", 800, 128.69)
Dim s3 As Stock = New Stock("LTWR", 1800, 12.95)
```

[end]

[C#]

```
Stock s1 = new Stock("AWRK",1200,28.47);
Stock s2 = new Stock("CTSO",800,128.69);
Stock s3 = new Stock("LTWR",1800,12.95);
```

[end]

6. Add these three instances to `stockValues`.

[Visual Basic]

```
stockValues.Add(s1)
stockValues.Add(s2)
stockValues.Add(s3)
```

[end]

[C#]

```
stockValues.Add(s1);
stockValues.Add(s2);
stockValues.Add(s3);
```

[end]

7. Add the updated stockValues ArrayList into session.

[Visual Basic]

```
Session("stockValues") = stockValues
```

[end]

[C#]

```
Session["stockValues"]=stockValues;
```

[end]

8. Inside of the Else block, write a line to assign the current values held in session to the `stockValues ArrayList`.

[Visual Basic]

```
stockValues = Ctype(Session("stockValues"), ArrayList)
```

[end]

[C#]

```
stockValues = (ArrayList)Session["stockValues"];
```

[end]

9. Finally, call the `PopulateStockValuesArrayList()` from the `ConfigureCrystalReports()` method. This should be the first line of code executed in the `ConfigureCrystalReports()` method.

[Visual Basic]

```
PopulateStockValuesArrayList()
```

[end]

[C#]

```
PopulateStockValuesArrayList();
```

[end]

To test the loading of the StockObjects report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start Debugging**.

   Note If this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   If no build errors appear, the Default.aspx page loads into your browser with three default values.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Populating the Object Collection Dynamically

In the previous section, you populated the Object Collection programmatically. In this section, you will learn how to dynamically add information to your data source from your website. This information will automatically update in your report.

To Add Controls to the Web Form

In this section, you add the controls to the Web Form that are needed to dynamically update the Object Collection.

1. Open the Default.aspx file in Design View.

   Note To open an ASPX page in design view, first open the file, and then click the **Design** button at the bottom of the form.
2. Click the **CrystalReportViewer** control to select it.
3. Press the left arrow key on your keyboard so that the flashing cursor appears, and the press enter several times.
4. From the **Toolbox** drag a **TextBox** control onto the web form.
5. From the **Property** menu, set the ID to "symbol."
6. Drag a second **TextBox** control onto the web form. Position the second **TextBox** below the first.
7. From the **Property** menu, set the ID to "price."
8. Drag a third **TextBox** control onto the web form. Position the third **TextBox** below the second.
9. From the **Property** menu, set the ID to "volume."

Note At this point, you may find it useful to add text beside each **TextBox** control to help identify which control corresponds to which parameter.

10. Next, from the **Toolbox**, drag a **Button** control onto the web form. Place the button below the three **TextBox** controls.
11. From the **Property** menu:

    Set the ID of the **Button** to "addStockInformation."

    Set the Text of the **Button** to "Add Stock Information."
12. Finally, click twice on the "Add Stock Information" button.

    Double clicking on the **Button** control will open up the Code-behind class and automatically generate an `addStockInformation_Click()` event handler.

To Add Information to the Collection

In this section, you will write code for the `addStockInformation_Click()` event handler that will add the information entered in the Web Form to the `stockValues` collection.

1. Inside of the `addStockInformation_Click()` event handler, create and instantiate a new Stock object.

[Visual Basic]

```
Dim temp As Stock = New Stock()
```

[end]

[C#]

```
Stock temp = new Stock();
```

[end]

2. Within the `addStockInformation_Click()` method, create a try/catch block.

[Visual Basic]

```
Try


Catch ex As Exception


End Try
```

[end]

[C#]

```
try
{
}
catch
{
}
```

[end]

Information entered into a web form is of type String. Because two of the fields within the Stock class are numerical, you will need to write code to convert the String values

from the web form into numerical values. The try/catch statement helps protect your web application from crashing during the conversion due to type mismatch errors.

3. Within the Try block, assign the value of the symbol field on the web form to the Symbol property of the Stock Object.

[Visual Basic]

```
temp.Symbol = symbol.Text
```

[end]

[C#]

```
temp.Symbol = symbol.Text;
```

[end]

4. On the next line, assign the value of the price field on the web form to the Price property of the Stock Object. Convert the value from the web form to a Double before assignment.

[Visual Basic]

```
temp.Price = CType(price.Text, Double)
```

[end]

[C#]

```
temp.Price = Convert.ToDouble(price.Text);
```

[end]

5. Next, assign the value of the volume field on the web form to the Volume property of the Stock Object. Convert the value from the web form to an integer before assignment.

[Visual Basic]

```
temp.Volume = CType(volume.Text, Integer)
```

[end]

[C#]

```
temp.Volume = Convert.ToInt32(volume.Text);
```

[end]

6. Outside of the try/catch block, add the Stock Object to the `stockValues ArrayList`.

[Visual Basic]

```
stockValues.Add(temp)
```

[end]

[C#]

```
stockValues.Add(temp);
```

[end]

7. Update the value of `stockValues` currently held in Session.

[Visual Basic]

```
Session("stockValues") = stockValues
```

[end]

[C#]

```
Session["stockValues"] = stockValues;
```

[end]

8. Finally, call the `ConfigureCrystalReports()` method. This will re-bind the report to the updated `stockValues` Object Collection.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

9. From the **Build** menu, click **Build Solution**.

   If you have any build errors, fix them now.

10. From the **Debug** menu, click **Start Debugging**.

    If no build errors appear, the Default.aspx page loads into your browser with three default values. To add additional values, fill in the TextBox Controls as appropriate and click the Add Stock Information button. The Report will dynamically update.

When you run your web site, the report will load in your browser window with the three default values that you programmatically added in Populating the Object Collection Programmatically. Above the report are three TextBox controls and a Button control. With these controls, you can dynamically update your Object Collection, and see the result of this update reflected in your report.

Conclusion

You created a unique class to hold stock market values, instantiated the class, populated an object collection with data, and dynamically added further data through a web form. You then created a Crystal report with the Crystal Report Designer control that connected to the object collection and dynamically generated a chart and stock summary.

Object collections are a versatile data source for your Crystal reports. With an Object collection, you can write a custom class that populates an object with data either programmatically or dynamically. Object collections can also be used to extend the number of data sources that are provided with a default installation Crystal Reports for Visual Studio by allowing you to write your own data access objects.

To learn more about how to work with an object collection in a report, continue to Addendum: Enhancing Your Report.

Addendum: Enhancing Your Report

At this point, you have a fully functional web site that will display a Crystal report based off on Object Collection. The site displays information that is programmatically entered into an Object Collection, as well as information that is added dynamically at run time.

In this section, you will add two charts, a calculated field, and summary information.

To Add a Simple Chart to the Report

In this section, you will add a simple pie chart to your report that displays the number of shares of a stock proportional to all other shares.

1. From the solution explorer, open StockObjects.rpt.
2. From the **Crystal Reports** menu, select **Insert**, and click **Chart**.

3. In the **Chart Expert** dialog box, select a **Pie** chart.
4. Select the **Data** tab from **Report Fields**.
5. Select **Stock.Symbol** and click the top most Right Arrow to move the **Stock.Symbol** field to the **On Change Of** field.
6. Select **Stock.Volume** and click the bottom most Right Arrow to move the **Stock.Volume** field to the **Show Value(s)** field.
7. Click **OK**.

   A new Report Header section is created, and a chart object is added to this section.
8. From the **Debug** menu, click **Start Debugging**.

   If no build errors appear, the Default.aspx page loads into your browser.

To Add a Chart Based on a Formula Field

In this section, you will create a chart that reports off of aggregate information. You will first create a formula to calculate the worth of a particular holding, and then create a pie chart that displays the proportional worth of all of your holdings.

1. From the **Crystal Reports** menu, select **Report**, and click **Formula Workshop**.
2. In the **Formula Workshop** dialog, select **Formula Fields**.
3. Click the **New** button to create a new formula.
4. In the **Custom Function Name** dialog, enter "worth."
5. Click **Use Editor**.
6. Add code to multiply the value of the **price** field by the value of the **volume** field.

```
{Stock.Volume}*{Stock.Price}
```

7. Click **Save and Close**.
8. From the **Crystal Reports** menu, select **Insert**, and click **Chart**.
9. In the **Chart Expert** dialog box, select a **Pie** chart.
10. Click the **Data** tab.
11. Select **Stock.Symbol** and click the top most Right Arrow to move the **Stock.Symbol** field to the **On Change Of** field.
12. Select **Worth** and click the bottom most Right Arrow to move the **Worth** formula to the **Show Value(s)** field.
13. Click the **Text** tab.
14. Beside **Title**, clear the **Auto-text** checkbox.
15. Enter "Worth / Symbol" in the **Title** field.
16. Click **OK**.
17. A new Report Header section is created, and a Chart Object is added to this section.

> **Note** To re-position the objects within a Crystal report, drag them with your mouse and place them where you like. You can use the **Main Report Preview** button at the bottom of the form to display a preview of your report.

To Add Formula and Summary Fields to your Report

In this section, you add a formula field to your report as well as a summary field that will calculate the total value of your portfolio.

1. Expand the **Formula Fields** node of the **Field Explorer**
2. Drag the **worth** formula onto your report. Position this field within the Details section of your report.

   Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.

   This field will display the worth of each row. Use a summary field to display the total worth of your portfolio.
3. From the **Crystal Reports** menu, select **Insert**, and click **Summary**.
4. Within the Insert Summary dialog box:

   Select the **Worth** formula from the **Choose the Field to Summarize** field.

   Select **Sum** from the **Calculate this Summary** field.

   Select **Grand Total** from the **Summary Location** field.
5. Click **OK**.

   A summary field is added to the report.
6. From the **Debug** menu, click **Start Debugging**.

   If no build errors appear, the Default.aspx page loads into your browser.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_Data_ObjectCollection

Visual Basic Web Site: VB_Web_Data_ObjectCollection

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Other Tutorials

Crystal Reports

For Visual Studio 2005

Other Tutorial:

Configuring Multilingual Client Support

Configuring Multilingual Client Support

Introduction

In this tutorial, you learn how to configure multilingual client support in a Web Site or a Windows project.

Crystal Reports for Visual Studio 2005 includes support for multilingual Web and Windows clients, through dynamic localization.

Dynamic localization allows users to view ToolTips and other content of the CrystalReportViewer control in their preferred language.

Note For an introduction to the concepts of dynamic localization, see Appendix: Multilingual Client Support.

In this tutorial, you study the following:

How to configure the CrystalReportViewer control language resources on the Windows client or Web server.

How to configure client access to localization in various ways.

How to configure global or local settings.

Viewing the Report with Default Settings

To begin, you build and compile a Web Site or Windows project, and then view the default language settings for the CrystalReportViewer toolbar.

To test the default locale of a report

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   The Hierarchical Grouping report displays successfully.

   Note For the project to compile, you must also have added a sample report as specified in Appendix: Additional Setup Requirements.
4. Roll the mouse over the buttons on the **CrystalReportViewer** control toolbar.

   ToolTips are now displayed in the default locale.
5. Return to Visual Studio and click **Stop** to exit from debug mode.

You are now ready to set up custom localized resource files for custom languages on your machine.

Working with Default and Custom Language Resource Files

Crystal Reports for Visual Studio 2005 is designed to work with several default language resource files, and also any custom language resource files created and compiled by the developer.

The default language resource files include the following:

English (en)

French (fr)

German (de)

Spanish (es)

Italian (it)

Japanese (jp)

Korean (ko)

Simplified Chinese (zh-chs)

Traditional Chinese (zh-cht)

Note In Crystal Reports for Visual Studio 2005, the default resource files are not shipped with the product install, but can be downloaded from the Business Objects Web Site. For the download location, see Appendix: Useful Addresses at a Glance.

In addition to the default languages, you can create your own customized language resource files for any other language.

In this tutorial you will practice custom resource file compilation by working with an uncompiled folder containing the Spanish default language resource files. For this tutorial you will rename this folder to represent a custom language.

You begin by setting up a custom resource files directory to hold your custom resource files.

To set up the custom resources files directory

1. Go to the Appendix: Tutorials' Sample Code Directory and locate the folder "CrystalReportViewer_resource_files".
2. Copy this folder to the root directory of your hard drive.

   Note This procedure assumes that your root directory is on the C: drive.
3. In Windows Explorer, open the folder named "C:\CrystalReportViewer_resource_files".

   Locate the subfolder, `es`, containing uncompiled Spanish language strings as .txt files. While Spanish is already supplied as one of the default compiled languages, you will use this directory to practice created a custom resource file. To do this, you rename the Spanish directory to Romanian, a language that is not one of the default languages. You then test this directory as if it contained Romanian language strings.
4. Rename the `es` directory to `ro`.

   Note `ro` is the abbreviation for the Romanian language.

For this tutorial, you will use the following:

Default resources for three supplied languages:

German (de)

French (fr)

Italian (it)

Manually compiled resources for one custom language:

Romanian (ro)

Follow the instructions in the next section to compile the custom resource files for the `ro` language folder.

Compiling the Custom Resource Files

You are now ready to compile the resource files for the custom language subdirectory that you created in the previous section.

Note If you need to compile custom resource files for languages with non-Latin scripts such as Cyrillic, Chinese, Korean, Japanese, Hebrew, and Aramaic, you will need to configure environment locales for those languages before compilation. This ensures that the proper language scripts are available. For more information, see Reference: Configuring an Environment Locale.

To compile the resource files

1. Open the `ro` language subdirectory.

   Three resource files are displayed:

   CRWebFormViewer.txt

   SCRShared.txt

   Viewer.txt

   The first two files contain resource strings for the Web-based CrystalReportViewer control. The last file contains resource strings for the Windows-based CrystalReportViewer control.

2. Open Viewer.txt to view its contents.

   The localized strings for the CrystalReportViewer control are displayed in the current language.

   Note This Romanian language folder currently contains Spanish-language strings. For more information, see the previous step in this tutorial, Working with Default and Custom Language Resource Files.

3. Close Viewer.txt.
4. From the **Start** menu, go to **Programs**>**Visual Studio 2005>Visual Studio Tools>Visual Studio Command Prompt**.
5. Change directory to the resource files directory:

   ```
   cd c:\CrystalReportViewer_resource_files\
   ```

6. Change directory again to the `ro` subdirectory.

   ```
   cd ro
   ```

7. Run the resgen utility to compile the .txt files, following the syntax shown below.

   ```
   resgen /compile CRWebFormViewer.txt,CrystalDecisions.Web.resources
   ```

   ```
   resgen /compile SCRShared.txt,CrystalDecisions.Shared.resources
   ```

```
resgen /compile Viewer.txt,CrystalDecisions.Windows.Forms.resources
```

Note Do not put a space either before or after the comma.

8. Run the al.exe utility to create a DLL for each resources file.

```
al.exe /t:lib /embed:CrystalDecisions.Web.resources /culture:ro
/out:CrystalDecisions.Web.custom_resources.dll

al.exe /t:lib /embed:CrystalDecisions.Shared.resources /culture:ro
/out:CrystalDecisions.Shared.custom_resources.dll

al.exe /t:lib /embed:CrystalDecisions.Windows.Forms.resources
/culture:ro /out:CrystalDecisions.Windows.Forms.custom_resources.dll
```

Note The output name changes the extension `resources.dll` to `custom_resources.dll`.

9. Type `dir` to view the compiled DLLs.

   You have completed resource compilation for this subdirectory.

10. If you have other custom languages that you wish to compile, repeat this section for each language.

Note To save time. enter all six commands into a batch file. Remember that the batch file must be run from the Visual Studio Command prompt to access the correct environment variables.

11. When you are finished compiling custom resources, close **Visual Studio Command Prompt**.

In the next section, you configure your Web Site or Windows project to access your custom resources from either their global or custom location.

If you are building a Web Site, continue to Web: Configuring Global or Local Custom Resources.

If you are building a Windows project, continue to Windows: Configuring Global or Local Custom Resources.

Web: Configuring Global or Local Custom Resources

Your Web Site accesses language resources either globally (at the system level) or locally (from within the Web Site folder).

Custom language resources may be placed in either a global or local location. Default language resources are always available globally.

Note In Crystal Reports for Visual Studio 2005, the default resource files are not shipped with the product install, but can be downloaded from the Business Objects Web Site and installed. For the download location, see Appendix: Useful Addresses at a Glance.

To expose custom resources globally, you place your custom resources in a central location in the file directory from which they can be shared across multiple Web applications. To then access those global resources in a specific Web Site, you place a reference to the global resources' file directory path in the Web.config file. For a global

resource to load correctly, the name of the folder that contains the resource files must match the language locale.

To expose custom resources locally, you place the custom resources directly into the Web Site folder. For a local resource to load correctly, the culture of the resource files and the name of the folder that contains the resource files must match the language locale.

The global approach is more common with Web Sites because this approach services all Web Sites from a single location on the Web server, and prevents redundancy and any risk of discrepancy between versions.

To access global custom resources (more typical for Web Sites)

In the Web.config file, add a globalResourcePath key to the appSettings node.

```
<appSettings>
   <add key="globalResourcePath"
value="c:\CrystalReportViewer_resource_files\"/>
</appSettings>
```

To access local custom resources (less typical for Web Sites)

Copy the custom resources subdirectory (in this case, the `ro` subdirectory) from the folder `C:\CrystalReportViewer_resource_files\` to the Web Site folder.

> **Note** If you have added a globalResourcePath key as shown under global resources, the local resources are ignored. This is because global resources take precedence over local resources in the access hierarchy.

Your Web Site is now configured to access your custom language resources either globally or locally and to access your default language resources globally. However, no particular language has yet been specified for viewing, so at this point, the CrystalReportViewer control remains in the default locale.

In the next three sections, you learn about browser-based, page-based, and web-server-environment-based localization.

Web: Configuring Browser-Based Localization

In this section, you learn about the browser-based scenario of dynamic localization. In this scenario, a property of the CrystalReportViewer control determines that dynamic localization will be based on the language setting of the client browser.

Because localization is browser-based in this scenario, each user's browser may retrieve a different language for the CrystalReportViewer control. This scenario is the most flexible when a wide variety of client browsers with many different language settings are expected to connect to your Web Site.

To begin, you learn how to configure multiple language settings in your client browser.

To configure browser-based localization

1. Launch your browser.

   > **Note** This tutorial assumes use of Internet Explorer, but other HTML 4.0-compliant browsers are also compatible.

2. In Internet Explorer, from the **Tools** menu, click **Internet Options**.
3. In the **Internet Options** dialog box, on the **General** tab, click **Languages...**
4. In the **Language Preference** dialog box, click **Add...**

5. Add the following 4 languages:
   German (Austria) [de-at]
   French (France) [fr]
   Italian (Italy) [it]
   Romanian [ro]
6. Select **Italian (Italy) [it]** and click **Move Up**. Repeat until **Italian (Italy) [it]** is the top-most selection.
7. Click **OK**, click **OK** again, and then close your browser.
8. Open your Web Site in **Visual Studio 2005**, and then build and compile.
9. Roll the mouse over the buttons on the **CrystalReportViewer** toolbar.
   The ToolTip strings still display in English, ignoring your browser's language setting. You must configure a setting named 'UseBrowserLocale' in your project's Web.config file so that the CrystalReportViewer control will observe the browser locale when rendering the ToolTip string values.
10. Close the browser to exit debug mode.
11. Open the Web.config file.
12. Within the `<configuration>` tag, enter the following nested tags to set the UseBrowserLocale property to true.
    A Web.config file can only have one `configSections` block, and that block must be the first child of the `configuration` tag. If your Web.config file already includes a `configSections` block, then edit that section to include the `sectionGroup` and `section` tags as shown below.

```
<configSections>
  <sectionGroup name="businessObjects">
    <sectionGroup name="crystalReports">
      <section name="crystalReportViewer"
      type="System.Configuration.NameValueSectionHandler" />
    </sectionGroup>
  </sectionGroup>
</configSections>


<businessObjects>
  <crystalReports=>
    <crystalReportViewer>
      <add key="UseBrowserLocale" value="true"/>
    </crystalReportViewer>
  </crystalReports>
</businessObjects>
```

Note In previous versions of Crystal Reports, the CrystalReportViewer properties included a UseBrowserLocale property. This property is now deprecated, and replaced with the above setting in the Web.config file. If no entry is placed in the Web.config, the UseBrowserLocale defaults to false.

13. Rebuild and compile your application.
14. Roll the mouse over the buttons on the **CrystalReportViewer** toolbar.

    The ToolTip strings now display in Italian. In the next step, you change the browser's language settings to German.
15. In Internet Explorer, from the **Tools** menu, click **Internet Options**.
16. In the **Internet Options** dialog box, on the **General** tab, click **Languages...**
17. Select **German (Austria) [de-at]** and click **Move Up**. Repeat until **German (Austria) [de-at]** is the top-most selection.
18. Click **OK**, and then click **OK** to return to your browser.
19. Click **Refresh**.
20. Roll the mouse over the buttons on the **CrystalReportViewer** toolbar.

    The ToolTip strings now display in German.
21. Repeat this process for **French (France) [fr]** and **Romanian [ro]**.

    In each case, the Browser language preferences determine which string is displayed in the CrystalReportViewer control.

    Note Remember! In this tutorial you have compiled your custom Romanian language resources based on language strings that were borrowed from a Spanish language directory, so your custom Romanian ToolTips will display in Spanish!
22. Return to Visual Studio and click **Stop** to exit from debug mode.

You have successfully configured dynamic localization using browser-based settings. In the next section, you learn how to configure localization using a setting on the ASPX page.

Web: Configuring Page-Based Localization

In this section, you learn about the page-based scenario of dynamic localization. In this scenario, a Page directive of the Default.aspx page determines dynamic localization.

Because localization is page-based in this scenario, each page in a Web Site could display a different language and bypass the language settings in the browser locale. So, this scenario is useful when you want to restrict the locale that is displayed on a page-by-page basis.

To begin, you learn how to reset the UseBrowserLocale property that you configured in the previous section.

To change the dynamic localization to be page-based

1. Open the Web.config file.
2. Within the `<configuration>` tag, enter the following nested tags to set the UseBrowserLocale property to false.

```
<configSections>
  <sectionGroup name="businessObjects">
    <sectionGroup name="crystalReports">
```

```
        <section name="crystalReportViewer"
          type="System.Configuration.NameValueSectionHandler" />
      </sectionGroup>
    </sectionGroup>
  </configSections>

  <businessObjects>
    <crystalReports=>
      <crystalReportViewer>
        <add key="UseBrowserLocale" value="false"/>
      </crystalReportViewer>
    </crystalReports>
  </businessObjects>
```

Note In previous versions of Crystal Reports, the CrystalReportViewer properties included a UseBrowserLocale property. This property is now deprecated, and replaced with the above setting in the Web.config file. If no entry is placed in the Web.config, the UseBrowserLocale defaults to false.

3. Open the **Default.aspx** page in **Design** view.
4. Click the **Default.aspx** page to select it.
5. From the **Properties** window, set the **Culture** property to **it-IT**.

   Note If you do not see the **Culture** property, select "Document" from the list at the top of the Properties windows.
6. Rebuild and compile your application.
7. Roll the mouse over the buttons on the **CrystalReportViewer** toolbar.

   The ToolTip strings display in Italian.
8. Return to Visual Studio and click **Stop** to exit from debug mode.

In this scenario, the following settings have been configured:

The CrystalReportViewer.UseBrowserLocale property has been set to False. Therefore, the browser locale setting, which was set to Spanish in the previous step procedure, is ignored.

The Page's Culture directive is set to Italian.

Therefore, the ToolTip strings display in Italian.

You have successfully configured dynamic localization with page-based settings. In the next section, you learn how to configure localization with the environment locale of the Web server.

Web: Configuring Environment-Based Localization

In this section, you learn about the Web-server-environment-based scenario of dynamic localization. In this scenario, environment locale settings for the Web server determine dynamic localization.

Because localization is environment-based, in this scenario the entire Web Site displays in a single language. So, this scenario is useful when you want to restrict the locale that is displayed for the entire Web Site to the environment locale that is configured on the Web server.

> **Note** The Environment locale is the default setting for the Web Site. If you choose to configure the Culture directive of a particular Page or the UseBrowserLocale property of a particular CrystalReportViewer control in your Web site, those configurations will override the default Environment locale setting and give you more customized localization at the Page or CrystalReportViewer control level.

To begin, you learn how to cancel the Culture Page directive that is configured in the previous section.

To configure environment-based localization

1. Open the **Default.aspx** page in **Design** view.

   "Document" is displayed at the top of the Properties window.
2. From the **Properties** window, clear the **Culture** property.
3. On your development machine (which is your Web server for this tutorial), change the default Environment locale in the **Regional Options** control panel to **German (Austria)**.

   > **Note** For detailed instructions on how to configure the Environment locale correctly in either Windows XP or Windows 2000, see the section at the end of this tutorial: Reference: Configuring an Environment Locale.
4. If you need to restart your machine to apply the regional settings, do so now.
5. Launch your Web Site again in **Visual Studio 2005**, then build and compile.
6. Roll the mouse over the buttons on the **CrystalReportViewer** toolbar.
7. When you have confirmed the language that is displayed, close the browser to exit debug mode.

In this scenario:

The CrystalReportViewer.UseBrowserLocale property has been set to False (this was performed earlier in this tutorial). Therefore, the browser locale setting, which was last set to Spanish, is ignored.

The Page Culture setting has been cleared. Therefore, no direction is given at the Page level.

The localization choice defaults to the Web server's environment locale, which you have just configured to "German (Austria)".

Therefore, the ToolTip strings display in German.

This completes the demonstration of multilingual client support with dynamic localization in Web Sites. The remaining sections demonstrate dynamic localization in Windows projects.

Windows: Configuring Global or Local Custom Resources

Your Windows project accesses language resources either globally (at the system level) or locally (from within the Windows executable folder).

Custom language resources may be placed in either a global or local location. Default language resources are always available globally.

> **Note** In Crystal Reports for Visual Studio 2005, the default resource files are not shipped with the product install, but can be downloaded from the Business Objects Web Site and installed. For the download location, see Appendix: Useful Addresses at a Glance.

To expose custom resources globally, you place your custom resources in a central location in the file directory from which they can be shared across multiple Windows applications. To then access those global resources in a specific Windows application, you place a reference to the global resources' file directory path in an XML configuration file stored in the same directory as the Windows executable. For a global resource to load correctly, the name of the folder that contains the resource files must match the language locale.

To expose custom resources locally, you place the custom resources into the Windows application at the same level as the executable file. For a local resource to load correctly, the culture of the resource files and the name of the folder that contains the resource files must match the language locale.

For deployment, your default language resources need to be included with the deployment. These resources will be accessed globally. For your custom resources, it is simpler to access them locally by configuring them to be installed in the same folder as the Windows executable.

In this section, you learn how to access local resources.

To access local custom resources

1. Change your Regional Options settings to "Romanian".

   > **Note** For detailed instructions on how to configure the Environment locale correctly in either Windows XP or Windows 2000, see the section at the end of this tutorial: Reference: Configuring an Environment Locale.

2. Copy the `ro` language subdirectory from the folder "C:\CrystalReportViewer_resource_files\" to the folder that contains the Windows executable. During development, this will likely be the `\bin\debug\` folder.
3. Compile and run the Windows application.
4. Roll the mouse over buttons on the toolbar.

   The ToolTip strings display in Romanian (more accurately, since we compiled the custom Romanian resource with test strings in Spanish, the tool tips will actually display Spanish.)

   > **Note** If you have added a global config file that contains a globalResourcePath key, the local resources are ignored. This is because global resources take precedence over local resources in the access hierarchy.

In the next section, you learn how to access global resources.

To access global custom resources (less typical approach for Windows applications)

1. Change your Regional Options settings to "Romanian".

   Note For detailed instructions on how to configure the Environment locale correctly in either Windows XP or Windows 2000, see the section at the end of this tutorial: Reference: Configuring an Environment Locale.

2. In Windows Explorer, locate the subdirectory of your Windows project that contains the executable file.

   ```
   \[PROJECT_PATH]\bin\debug\
   ```

3. In this subdirectory, create a new XML configuration file with the following information:

   ```
   <?xml version="1.0"?>
   <configuration>
      <appSettings>
            <add key="globalResourcePath"
   value="c:\CrystalReportViewer_resource_files\"/>
      </appSettings>
   </configuration>
   ```

4. Save this file with the same name as the executable file, but add a .config extension.

   For example:

   Executable name: `CS_Win_Multilingual.exe`

   Config file name: `CS_Win_Multilingual.exe.config`

   Note In Visual Basic Windows applications, this file may have been automatically generated. In that case, append the `<app settings>` node within the existing `<configuration>` node in the auto-generated config file.

5. Compile and run the Windows application.
6. Roll the mouse over buttons on the toolbar.

   The ToolTip strings display in Romanian (more accurately, since we compiled the custom Romanian resource with test strings in Spanish, the tool tips will actually display Spanish.)

Conclusion

In this tutorial, you learned about the following procedures:

How to configure CrystalReportViewer control language resources on the Windows client or Web server in a wide variety of languages.

How to configure user access to localization in various ways.

You also examined alternative configurations for global or local settings and determined the most suitable approach for a Web Site or Windows project.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_Multilingual

C# Windows project: CS_Win_Multilingual

Visual Basic Web Site: VB_Web_Multilingual

Visual Basic Windows project: VB_Win_Multilingual

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Reference: Configuring an Environment Locale

This section applies to both Web Sites and Windows projects, for times when you need to change the environment locale. For this example, the regional settings are set to "German (Austria)".

The following instructions apply to Windows XP and Windows 2000.

To configure the environment locale in Windows XP

1. Click **Start**, click **Control Panel**, and then click **Regional and Language Options**.
2. On the **Regional Options** tab, from the **Select an item to match its preferences** list, select **German (Austria)**.
3. In the **Languages** tab, click **Details...**

   The Text Services and Input Languages dialog box is displayed.
4. In the **Settings** tab, click **Add...**
5. In the **Add Input Language** dialog box, do the following:

   Set the **Input Locale** list to **German (Austria)**.

   Set the **Keyboard layout/IME** to **German (IBM)**.
6. Click **OK**.
7. In the **Default input language** list, select **German (Austria)**, and then click **OK**.
8. In the **Languages for non-Unicode programs** list, select **German (Austria)**.
9. Click **Apply**, and then click **OK**.

In Windows XP, you do not need to reboot your computer for these changes to be applied.

To configure the environment locale in Windows 2000

1. Click **Start** menu, point to **Settings**, click **Control Panel**, and then click **Regional Options**.

   The Regional Options control panel is displayed.
2. on the **General** tab, do the following:

   Click the **Your locale (location)** list and select **German (Austria)**.

Click **Set Default...**

In the **Select System Locale** dialog box select **German (Austria)**, and then click **OK**.

3. In the **Input Locales** tab, do the following:

   Click **Add...**

   In the **Add Input Locale** dialog box, set the **Input Locale** list to **German (Austria)**.

   Set the **Keyboard layout/IME** to **German (IBM)**, and then click **OK**.

4. Click **Apply**.

   A dialog box is displayed, and it asks you to install files either from your hard drive, or from the Windows Installer CD ROM.

5. Click **Yes** and proceed with the installation.

   When installation is complete, in the Change Regional Options dialog box you are asked to restart your computer.

6. Click **Yes**.

In Windows 2000, you must reboot your computer for these changes to be applied.

Crystal Reports

For Visual Studio 2005

Other Tutorial:

Creating a User Function Library

Creating a User Function Library

Introduction

In this tutorial, you learn how to create a User Function Library that can be accessed from the Formulas Expert in the embedded Crystal Report Designer.

When developers create a Crystal report in the embedded Crystal Report Designer, they often create formula fields to use in the report. These formula fields consist of variables and functions. The embedded Crystal Report Designer comes with a large collection of existing functions.

However, in some scenarios a user may require the ability to create a customized function, in the form of a library that can be distributed across machines. The User Function Library provides this capability.

A User Function Library (UFL) is a .NET class library that has the following characteristics:

The project name is prefixed with the string "CRUFL".

The project consists of an interface and an implementation class.

Both the interface and the implementation class have Com and GUID attributes.

Upon compilation, the assembly is registered for COM Interop so that it is viewed as a COM object.

The assembly/COM object is added to the Global Assembly Cache so that it is available across all .NET applications on the machine.

This assembly/COM object (the User Function Library, or UFL) then becomes visible within the embedded Crystal Reports Designer as a customized function under "Functions>Additional Functions>u2lcom.dll".

To begin, you create the class library.

Creating the UFL Class Library

In this section, you create the UFL Class library and configure it to be registered for COM Interop.

To create a UFL class library

1. Launch Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Project**.
3. In the **New Project** dialog box, select a language folder for C# or Visual Basic from the **Project Types** list.
4. From the **Templates** list, click **Class Library**.
5. In the **Name** field, enter one of the following:

   `CRUFL_VB_ExchangeRate` (Visual Basic)

   `CRUFL_CS_ExchangeRate` (C#)

   Note All User Function Libraries must be prefixed with `CRUFL` to be recognized by the embedded Crystal Reports Designer.
6. Click **OK**.
7. In **Solution Explorer**, right-click the project name that is in bold type, and then click **Properties**.

8. Click the **Build/Compile** tab.
9. If you are using Visual Basic, on the **OptionStrict** list, click **On**.
10. Scroll to the bottom of the **Build/Compile** window and select **Register for COM Interop**.

    Note "Register for COM Interop" causes the assembly to be automatically registered by the regasm.exe utility, when the project is compiled. So, you do not need to run regasm.exe manually from the command prompt.
11. From the **File** menu, click **Save All**.
12. Close the **Compile/Build** window.

You are now ready to create the interface.

Creating the Interface with COM Attributes

In this section, you create an interface and configure it with COM attributes.

To create the interface

1. Delete the default class (Class1).
2. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add**, and then click **Class**.
3. In the **Name** field, enter "IExchangeUfl", and then click **Add**.
4. Change the class signature from class to interface, and set the scope to public.

    Note In Visual Basic, remember to change both the opening and closing signature. In C#, delete the constructor.

[Visual Basic]

```
Public Interface IExchangeUfl

End Interface
```

[end]

[C#]

```
public interface IExchangeUfl

{

}
```

[end]

5. Above the class declaration, add the following using/Imports statement:

[Visual Basic]

```
Imports System.Runtime.InteropServices
```

[end]

[C#]

```
using System.Runtime.InteropServices;
```

[end]

6. From the **Tools** menu, click **Create GUID**.
7. In the **Create GUID** dialog box, in the **GUID format** panel, select **Registry Format**.
8. Click **New GUID**.
9. Click **Copy**, and then close the dialog box.

10. Above the interface declaration, create an attribute with three values: ComVisible, InterfaceType, and Guid (in C#) or GuidAttribute (in Visual Basic) with parameter string quotes.

    Note In Visual Basic, for readability add an underscore after the close tag to place the class on the next line.

[Visual Basic]

```
<ComVisible(), InterfaceType(), GuidAttribute("")> _
Public Interface IExchangeUfl
End Interface
```

[end]

[C#]

```
[ComVisible(), InterfaceType(), Guid("")]
public interface IExchangeUfl
{
}
```

[end]

11. Enter a parameter value of "True" [Visual Basic] or "true" [C#] into the ComVisible attribute.
12. Enter a parameter value of "ComInterfaceType.InterfaceIsDual" from the ComInterfaceType enum into the InterfaceType attribute.
13. Paste the GUID value from the clipboard into the parameter value of the Guid attribute. Be careful to remove the curly braces and any carriage returns.

    Note Do not use the GUID provided in the code snippet below. Create a unique GUID for your interface.

[Visual Basic]

```
<ComVisible(True), InterfaceType(ComInterfaceType.InterfaceIsDual),
GuidAttribute("E7A4EC98-BF2B-4006-B266-74C74421C394")> _
Public Interface IExchangeUfl
End Interface
```

[end]

[C#]

```
[ComVisible(true), InterfaceType(ComInterfaceType.InterfaceIsDual),
Guid("E7A4EC98-BF2B-4006-B266-74C74421C394")]
public interface IExchangeUfl
{
}
```

[end]

14. Within the IExchangeUfl interface, create a method signature.

    Note This method signature will become the name of the function that is exposed in the embedded Crystal Reports Designer.

[Visual Basic]

```
Function ConvertUSDollarsToCDN(ByVal usd As Double) As Double
```

[end]

[C#]

```
double ConvertUSDollarsToCDN(double usd);
```

[end]

15. Close the interface window.

Next you create the implementation class.

Creating the Implementation Class with COM Attributes

In this section, you create an implementation class that implements the IExchangeUfl interface, and then configure that class with COM attributes.

To create the implementation class

1. In **Solution Explorer**, right-click the bold project name that is in bold type, point to **Add**, and then click **Class**.
2. In the **Name** field, enter "ExchangeUfl", and then click **Add**.
3. Modify the class signature to implement the IExchangeUfl interface.

[Visual Basic]

```
Public Class ExchangeUfl : Implements IExchangeUfl

End Class
```

[end]

[C#]

```
public class ExchangeUfl : IExchangeUfl
{
}
```

[end]

4. Above the class declaration, add the following using/Imports statement:

[Visual Basic]

```
Imports System.Runtime.InteropServices
```

[end]

[C#]

```
using System.Runtime.InteropServices;
```

[end]

5. From the **Tools** menu, click **Create GUID**.
6. In the **Create GUID** dialog box, in the **GUID format** panel, select **Registry Format**.
7. Click **New GUID**.
8. Click **Copy**, and then close the dialog box.

9. Above the class declaration, create an attribute with three values: ComVisible, ClassInterface and Guid (in C#) or GuidAttribute (in Visual Basic) with parameter string quotes.

   Note In Visual Basic, for readability add an underscore after the close tag to place the class on the next line.

[Visual Basic]

```
<ComVisible(), ClassInterface(), GuidAttribute("")> _
Public Class ExchangeUfl : Implements IExchangeUfl
End Class
```

[end]

[C#]

```
[ComVisible(), ClassInterface(), Guid("")]
public class ExchangeUfl : IExchangeUfl
{
}
```

[end]

10. Enter a parameter value of "True" [Visual Basic] or "true" [C#] into the ComVisible attribute.
11. Enter a parameter value of "ClassInterfaceType.None" from the ClassInterfaceType enum into the ClassInterface attribute.
12. Paste the GUID value from the clipboard into the parameter value of the Guid attribute. Be careful to remove the curly braces and any carriage returns.

    Note Do not use the GUID that is provided in this code snippet. Create a unique GUID for your class.

[Visual Basic]

```
<ComVisible(True), ClassInterface(ClassInterfaceType.None),
GuidAttribute("F5DCE88F-AD38-4a9a-9A69-0F8DC0EDB4E3")> _
Public Class ExchangeUfl : Implements IExchangeUfl
End Class
```

[end]

[C#]

```
[ComVisible(true), ClassInterface(ClassInterfaceType.None),
Guid("F5DCE88F-AD38-4a9a-9A69-0F8DC0EDB4E3")]
public class ExchangeUfl : IExchangeUfl
{
}
```

[end]

13. Within ExchangeUfl, fulfill the contractual method signature from the interface with a public method.

Note This will be the name of the function that is exposed in the embedded Crystal Reports Designer.

[Visual Basic]

```
Public Function ConvertUSDollarsToCDN1(ByVal usd As Double) As Double
Implements IExchangeUfl.ConvertUSDollarsToCDN
End Function
```

[end]

[C#]

```
public double ConvertUSDollarsToCDN(double usd)
{
}
```

[end]

14. Within the method, create a conditional block that checks whether the usd method parameter is greater than Double.MaxValue.

Note You need to perform this check to prevent the risk of an overflow.

[Visual Basic]

```
If usd > Double.MaxValue Then
End If
```

[end]

[C#]

```
if(usd > Double.MaxValue)
{
}
```

[end]

15. Within the If block, throw an exception that states the value submitted is larger than the maximum value allowed for a double.

[Visual Basic]

```
Throw New Exception("Value submitted is larger than the maximum value
allowed for a double.")
```

[end]

[C#]

```
throw new Exception("Value submitted is larger than the maximum value
allowed for a double.");
```

[end]

16. Following the conditional block, return the usd method parameter multiplied by an exchange rate of 1.45.

[Visual Basic]

```
Return (usd * 1.45)
```

[end]

[C#]

```
    return (usd * 1.45);
[end]
```

> **Note** In this tutorial, only a simple math calculation is performed. However, any code could be placed within this method. For example, you could retrieve an exchange rate from a bank web service, to calculate the exchange rate dynamically within your report.

17. From the **File** menu, click **Save All**.
18. Close the class window.

In the next section, you assign a strong name key to the library, and then compile the project.

Assigning a Strong-Name Key to the Class Library

In this section, you assign a strong-name key to the class library.

To assign a strong-name key

1. In **Solution Explorer**, right-click the bold project name that is in bold type, and then select **Properties**.
2. Click the **Signing** tab.
3. Select the **Sign the assembly** checkbox.
4. From the **Choose a strong name key file:** combo box, select **<New...>**
5. In the **Create Strong Name Key** dialog box, in the **Key file name:** text field, enter a string value.

   > **Note** For example, a possible string value could be your last name and today's date.

6. Clear **Protect my key file with a password**, and then click **OK**.
7. Close the **Signing** window.
8. From the **Build** menu, click **Build Solution**.
9. From the **File** menu, click **Save All**.
10. Close Visual Studio 2005.

In the next section, you add the assembly to the Global Assembly Cache (the GAC).

Adding the Assembly to the Global Assembly Cache

In this section, you add the assembly to the Global Assembly Cache (GAC).

To add the assembly to the Global Assembly Cache

1. In Windows Explorer, locate the project in \My Documents\Visual Studio 2005\Projects\.
2. Within the project that you have created, locate the subdirectory \bin\Debug\.
3. Confirm that the assembly DLL is located in this directory.
4. Copy the file directory path of the DLL to the clipboard.

5. Click **Start**, point to **Programs**>**Microsoft Visual Studio 2005>Visual Studio Tools**, and then click **Visual Studio Command Prompt**.

   You need to change directory to the directory containing your DLL.

6. To do this, begin by typing the change directory command:

   ```
   cd
   ```

7. Add a space after the cd command, and then right-click to paste the directory file path from the clipboard.
8. Press **Enter**.

   Verify that the directory has changed to the directory that contains the assembly DLL.

9. Type the following command, and replace the assembly DLL shown in the code snippet with the name of your assembly DLL.

   ```
   gacutil -if CRUFL_CS_ExchangeRate.dll
   ```

10. Press **Enter**.
11. The following confirmation message is displayed:

    ```
    Assembly successfully added to the cache.
    ```

12. If a failure message appears, recheck the spelling of the assembly DLL filename and run `gacutil` again.
13. Close **Visual Studio Command Prompt**.
14. In Windows Explorer, locate the subdirectory \assembly\, which is within the \Windows\ or \WinNT\ directory.

    The assembly you added should now be visible in the assembly subdirectory (known as the GAC, or Global Assembly Cache.)

You are now ready to create a Crystal report that accesses this custom function.

Adding the Custom Function to a Crystal Report

In this section, you create a Crystal report that accesses your custom function.

To add the custom function to a Crystal report

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. In **Solution Explorer**, right-click the project name in bold type, point to **Add**, and then click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** view, select the template named "Crystal Report."
3. In the **Name** field, enter the name "FunctionTest.rpt", and then click **Add**.
4. If you have not registered before, you are asked to register. To learn how to register, see Appendix: Crystal Reports Registration and Keycode.
5. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.

6. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
7. In the **Available Data Sources** panel of the Standard Report Creation Wizard window, expand the **Create New Connection** folder.
8. From the subfolder that opens, expand the **ODBC (RDO)** folder.
9. In the ODBC (RDO) window, select the correct ODBC DSN entry for your version of Crystal Reports, as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

   The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
10. Expand the **Tables** node and select the **Customer** table.
11. Double-click the **Customer** table to move the table into the **Selected Tables** panel, and then click **Next**.
12. CTRL-click **Customer Name**, **Last Year's Sales**, and **City**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click **Next**.
14. In the **Available Fields** panel, under **Report Fields**, select **Customer.City**, click the **>** symbol to move the field into the **Group By** panel, and then click **Finish**.

    The FunctionTest report is created and loaded into the main window of Visual Studio.
15. If the **Field Explorer** is not visible, on the Crystal Reports toolbar, click **Toggle Field View**.

    Note Another way to display the **Field Explorer** is to go to the **Crystal Reports** menu, and then click **Field Explorer**.
16. In the **Field Explorer**, right-click **Formula Fields**, and then click **New...**
17. In the **Formula Name** dialog box, type the name "MyFunctionFormula", and then click **Use Editor**.

    The Formula Workshop window is displayed, showing four panels horizontally.
18. In the third panel, expand **Functions**.
19. Expand **Additional Functions**, and within it expand **Visual Basic UFLs (u2lcomm.dll)**.

    The function that you created is displayed: ConvertUSDollarsToCDN.
20. Double-click **ConvertUSDollarsToCDN**.

    The ConvertUSDollarsToCDN() function appears in the lower panel, where you can now create your formula.
21. In the lower panel, place your cursor within the function brackets.
22. In the second panel, expand **Report Fields**, and then double-click **Customer.Last Year's Sales**.

    Customer.Last Year's Sales is entered as the parameter within the ConvertUSDollarsToCDN function.
23. On the toolbar, click **Check**.

    The formula is verified, with no errors found.
24. Close the **Formula Editor**.

25. From the **Field Explorer**, within the **Formula Fields** node, drag **MyFunctionFormula** to the right of the **Last Year's Sales** field in the **Detail** section of the report.

    Arrange the fields to fit correctly on the report.

26. Click **Main Report Preview**.

    The report is displayed, with your custom function having calculated the value of the MyFunctionFormula field.

27. Click **Main Report** to exit Preview mode.

You are now ready to bind and display your report.

Binding the Report

In Appendix: Project Setup, you placed a CrystalReportViewer control on the Web or Windows Form. In the previous step, you have added a FunctionTest report to the project.

In this section, you instantiate the FunctionTest report and bind it to the CrystalReportViewer control.

You can instantiate and bind the report in two ways:

As an embedded report.

As a non-embedded report.

Note Visual Studio 2005 supports only non-embedded reports for Web Sites.

Choose from one (but not both) of the step procedures below.

If you use embedded reports, follow the next step procedure to instantiate the report as an embedded report.

If you use non-embedded reports, follow the second step procedure to instantiate the report as a non-embedded report.

To instantiate the FunctionTest report as an embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the FunctionTest report wrapper class, using the variable name functionTestReport. Set its access modifier to private.

[Visual Basic]

```
Private functionTestReport As FunctionTest
```

[end]

[C#]

```
private FunctionTest functionTestReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method, instantiate the report wrapper class.

   Note You created the `ConfigureCrystalReports()` method in Appendix: Project Setup.

[Visual Basic]

```
functionTestReport = New FunctionTest()
```

[end]

[C#]

```
functionTestReport = new FunctionTest();
```

[end]

5. On the next line beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer control to the instantiated report class (variable name: functionTestReport).

[Visual Basic]

```
myCrystalReportViewer.ReportSource = functionTestReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = functionTestReport;
```

[end]

You are now ready to build and run your project. Skip to the next section below.

To instantiate the FunctionTest report as a non-embedded report and bind it to the CrystalReportViewer control

1. Open the Web or Windows Form.
2. From the **View** menu, click **Code**.
3. Add a new class-level declaration for the ReportDocument report wrapper class, using the variable name functionTestReport. Set its access modifier to private.

[Visual Basic]

```
Private functionTestReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument functionTestReport;
```

[end]

> **Note** The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You have added an `"Imports" [Visual Basic]` or `"using" [C#]` declaration for this namespace in Appendix: Project Setup. When you instantiate ReportDocument and load a report into the namespace, you gain access to the report through the SDK, without embedding the report.

4. Within the `ConfigureCrystalReports()` method (that you have created in Appendix: Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
functionTestReport = New ReportDocument()
```

[end]

[C#]

```
functionTestReport = new ReportDocument();
```

[end]

5. In the next line, call the `Load()` method of the ReportDocument instance and paste into it the report file name.

Note Wrap the report file name in a `Server.MapPath()` method. This will automatically generate the file directory path to the report based on the report name.

[Visual Basic]

```
functionTestReport.Load(Server.MapPath("FunctionTest.rpt"))
```

[end]

[C#]

```
functionTestReport.Load(Server.MapPath("FunctionTest.rpt"));
```

[end]

6. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = functionTestReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = functionTestReport;
```

[end]

Whether you have chosen to instantiate an embedded report class or a non-embedded report class (ReportDocument), the variable name used is the same: functionTestReport. This allows you to use a common set of code in the procedures that follow.

You are now ready to build and run your project.

To test the loading of the FunctionTest report

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The report displays, with the new field calculating an exchange rate for last year's sales, based upon the User Function Library that you created.
4. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

In this tutorial, you built a User Function Library by creating a .NET class library, populating it with an interface and implementation class, setting COM attributes on both interface and implementation class, and registering the assembly for COM Interop. You built the assembly, added it to the Global Assembly Cache, and then accessed it within the embedded Crystal Report Designer to apply your custom function within a formula field.

You then bound the report and displayed it at runtime, where you saw the new conversion field calculating a value based upon the User Function Library that you created.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Class Library: CRUFL_CS_ExchangeRate

VB Class Library: CRUFL_VB_ExchangeRate

To locate the folders that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Other Tutorial:

Populating a Drop Down List of Reports from the File Directory

Populating a DropDown List of Reports from the File Directory

Introduction

In this tutorial, you learn how to populate a DropDownList control with a list of non-embedded reports located on the local file system.

It is common to hard code the path to a non-embedded report from your web application. A hard coded file path is easy to work with, but increases maintenance time. Hard coded file paths require that you rebuild the project each time you add a report to your application, or change an existing report.

In this tutorial, you learn how to populate a DropDownList control with a list of reports from the local file system. This allows you to add or change a report without having to rebuild the project.

Adding Controls to the Web Form

In this section, you add the DropDownList control and Button control to the web form.

> **Note** This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

To create and configure a DropDownList Control on the Form

1. Open the Web form.
2. From the **View** menu, click **Designer**.
3. Click the **CrystalReportViewer** control to select it.
4. Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER.

   The CrystalReportViewer control drops by one line.
5. From the **Toolbox**, drag a **DropDownList** control above the **CrystalReportViewer** control.

   > **Note** If a Smart Task appears on the DropDownList (when using Visual Studio 2005), press Esc to close it.
6. Click on the **DropDownList** control to select it.
7. From the **Properties** window, set the **ID** to "reportsList"
8. From the **File** menu, click **Save All**.

You are now ready to add a button control to the web form.

To create a Button Control on the Web Form

1. Open the Web form.
2. From the **View** menu, click **Designer**.
3. From the **Toolbox**, drag a **Button** control to the right of the **DropDownList** control.
4. Click the **Button** control to select it.

5. From the **Properties** window:

   Set the **ID** to "display"

   Set the **Text** to "Display Report"

In the next step, you write code to automatically populate the **DropDownList** control.

Populating the DropDownList Control

In this section, you write code to populate a DropDownList control with a list of non-embedded reports contained within a subdirectory.

> **Note** In this tutorial, you will populate the DropDownList control with reports contained within a subdirectory of your web site. You can also address folders elsewhere on your file system. If you already have a folder that contains several reports, you can skip the next procedure and move directly to **Populating the DropDownList Control**.

To create a Reports folder

1. In **Solution Explorer**, right-click the project name that is in bold type, point to **Add Folder**, and then click **Regular Folder.**
2. Name this folder **Reports**.
3. In **Solution Explorer**, right-click the **Reports folder**, click **Add Existing Item**.
4. In the **Add Existing Item** dialog box, set **Files of type** to All Files (*.*).
5. Navigate to the **Hierarchical Grouping.rpt** file in the Feature Examples folder of the Crystal Reports sample reports directory. (See Appendix: Sample Reports' Directory for the locations of the sample reports.)

   > **Note** The Hierarchical Grouping report retrieves its data from the Access database xtreme.mdb. If you have not verified the location of this database and its ODBC DSN configuration, see Appendix: What Needs to be Verified?.

6. Click the **Hierarchical Grouping.rpt** file to select it, and then click **Open**.

   The Hierarchical Grouping.rpt file is added to your project.

7. Add two more reports from the Sample Reports directory before moving to the next procedure.

This completes the section on adding a Reports folder.

Populating the DropDownList Control

In this procedure, you populate a sorted list of the reports contained within the Reports subdirectory.

1. Open the Web Form.
2. From the **View** menu, click **Code**.
3. Above the class signature, add an `"Imports"[Visual Basic]` or `"using"[C#]` declaration to the top of the class for the System.IO and System.Collections namespaces.

[Visual Basic]

```
Imports System.IO
Imports System.Collections
```

[end]

[C#]

```
using System.IO;
using System.Collections;
```

[end]

4. At the class level, create a new `ReportDocument` variable, `reportDocument`.

[Visual Basic]

```
Private myReportDocument As ReportDocument
```

[end]

[C#]

```
private ReportDocument reportDocument;
```

[end]

5. Within the `page_init()` method, create a `Not IsPostBack` conditional block.

[Visual Basic]

```
If Not IsPostBack Then
Else
End If
```

[end]

[C#]

```
if(!IsPostBack)
{
}
else
{
}
```

[end]

Note The `Not IsPostBack` conditional block is used to encapsulate code that should only be run the first time the page loads. Controls are typically bound to data values within `Not IsPostBack` conditional blocks so that their data values (and any subsequent control events) are not reset during page reloads.

6. Within the `Not IsPostBack` conditional block, create a String variable that will hold the path to your reports folder.

[Visual Basic]

```
Dim myReportPath As String = Server.MapPath("Reports/")
```

[end]

[C#]

```
string reportPath = Server.MapPath("Reports/");
```

[end]

Note If you skipped the previous procedure, and you have an existing directory of reports, enter the file path to that directory in place of "Reports/".

7. Use the `Directory.GetFiles()` method to retrieve the contents of the Reports folder. Assign the result to the `reports` array.

[Visual Basic]

```
Dim reports() As String = Directory.GetFiles(myReportPath, "*.rpt")
```

[end]

[C#]

```
string[] reports = Directory.GetFiles(reportPath, "*.rpt");
```

[end]

8. On the next line, instantiate the `SortedList` class and pass it to its parent interface, `IDictionary`.

[Visual Basic]

```
Dim mySortedList As IDictionary = New SortedList
```

[end]

[C#]

```
IDictionary sortedList = new SortedList();
```

[end]

9. Next, create a `For Each` loop that loops through each element of `reports`.

[Visual Basic]

```
For Each path As String In reports
Next
```

[end]

[C#]

```
foreach (string path in reports)
{
}
```

[end]

10. The next three lines of code use string manipulation methods to remove the file path from the name of each report. Insert these lines inside of the `For Each` loop.

[Visual Basic]

```
Dim reportNamePrefix As Integer = path.LastIndexOf("\") + 1
Dim reportNameLength As Integer = path.Length - reportNamePrefix
Dim reportName As String = path.Substring(reportNamePrefix,
reportNameLength)
```

[end]

[C#]

```
int reportNamePrefix = path.LastIndexOf(@"\") + 1;
int reportNameLength = path.Length - reportNamePrefix;
string reportName = path.Substring(reportNamePrefix, reportNameLength);
```

[end]

11. Still inside the `For Each` loop, add a line of code that adds the truncated report name to the `sortedList` IDictionary.

[Visual Basic]

```
mySortedList.Add(path, reportName)
```

[end]

[C#]

```
sortedList.Add(path, reportName);
```

[end]

Binding the Sorted List to the DropDownList Control

In this section, you learn how to bind the `IDictionary` data source to the `DropDownList` control. First, you define a relationship between the elements of the `IDictionary` collection and the `DropDownList` fields. Then, you bind the `IDictionary` data source to the `DropDownList` control.

1. Outside of the `foreach` loop, add code to assign the Value field of the `DropDownList` to the key property of the `sortedList` instance.

[Visual Basic]

```
reportsList.DataTextField = "value"
```

[end]

[C#]

```
reportsList.DataTextField = "value";
```

[end]

2. Next, assign the Value field of the `DropDownList` control to the key property of the `sortedList` instance.

[Visual Basic]

```
reportsList.DataValueField = "key"
```

[end]

[C#]

```
reportsList.DataValueField = "key";
```

[end]

3. Assign the `SortedList` instance to the `dataSource` property of the `DropDownList` control.

[Visual Basic]

```
reportsList.DataSource = mysortedList
```

[end]

[C#]

```
reportsList.DataSource = sortedList;
```

[end]

4. Finally, bind the data source to the `DropDownList`.

[Visual Basic]

```
reportsList.DataBind()
```

[end]

[C#]

```
    reportsList.DataBind();
[end]
```

You have now completed the contents of the `If Not IsPostBack` conditional block. In the next section, you will write code inside of the `Else` conditional block to define what happens if the page is refreshed without assigning a new value to the `ReportDocument` instance.

5. Inside of the else statement, assign the report document instance to the report that is currently held in session.

[Visual Basic]

```
    myReportDocument = CType(Session("myReportDocument"), ReportDocument)
[end]
```

[C#]

```
    reportDocument = (ReportDocument)Session["reportDocument"];
[end]
```

> **Note** Because Session only returns generic objects, you must cast the report to a report type. Whether you use embedded or non-embedded reports, cast the retrieved object to the ReportDocument type.

6. On the next line, bind the ReportDocument instance to the CrystalReportViewer control.

[Visual Basic]

```
    myCrystalReportViewer.ReportSource = myReportDocument
[end]
```

[C#]

```
    crystalReportViewer.ReportSource = reportDocument;
[end]
```

7. From the **File** menu, select **Save All**.
8. From the **Build** menu, select **Build Solution**.
9. If you have any build errors, go ahead and fix them now.

In the next section, you add a click event to bind the report based on selections from the reportsList DropDownList control.

Adding a Click Event to Bind the Report

In this section you configure the display Button control to display the report selected from the reportsList control. Within the event method for this button, you rebind the CrystalReportViewer control to the selected report.

To create the click event method for the Button in a Web project

1. Open the Web form.
2. From the **View** menu, click **Designer**.
3. Double-click the display Button control.

   You are taken to the code-behind class where a `display_Click()` event method has been automatically generated.

You are now ready to create the helper method that binds the value of `reportsList` to the `crystalReportViewer` control.

4. Within the `display_Click()` event method that has just been auto-generated, instantiate a new `ReportDocument`.

[Visual Basic]

```
myReportDocument = New ReportDocument
```

[end]

[C#]

```
reportDocument = new ReportDocument();
```

[end]

5. Call the `Load()` method of the ReportDocument instance and pass it the value selected from `reportsList`.

[Visual Basic]

```
myReportDocument.Load(reportsList.SelectedValue)
```

[end]

[C#]

```
reportDocument.Load(reportsList.SelectedValue);
```

[end]

6. Assign the report into Session, using the report variable name as the Session identifier string.

[Visual Basic]

```
Session("myReportDocument") = myReportDocument
```

[end]

[C#]

```
Session["reportDocument"] = reportDocument;
```

7. Rebind the report instance to the ReportSource property of the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = myReportDocument
```

[end]

[C#]

```
crystalReportViewer.ReportSource = reportDocument;
```

[end]

8. From the **File** menu, select **Save All**.
9. From the **Build** menu, select **Build Solution**.
10. If you have any build errors, go ahead and fix them now.

Running the Application

1. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

2. Select a report from the DropDownList
3. Click **Display Report** to display the report that you selected.

Conclusion

In this tutorial you learned how to populate a DropDownList control with a list of Crystal reports located on the local file system.

First, you created a web form and added a Button control, a DropDownList control and a CrystalReportViewer control. Then, you populated the DropDownList control with a list of reports located on the local file system. Finally, you created a click event method that binds the report selected from the DropDownList control to the CrystalReportViewer control.

It is also possible to populate a list of reports from a remote server using Crystal Reports web services.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_DrpFileDir

Visual Basic Web Site: VB_Web_DrpFileDir

To locate the folders that contain these samples, see Appendix: Tutorials Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Other Tutorial:

Populating a Drop Down List of Reports from a Web Service

Populating a DropDown List of Reports from a Web Service

Introduction

With Crystal Reports for Visual Studio, you can publish Crystal Reports as Web services.

In this tutorial, you learn how to use Crystal Reports web services to populate a DropDownList control with a list of non-embedded reports from a web service.

Crystal Reports has enabled reports to be easily published and consumed as Report Web Services. Web and Windows applications can connect to a Report Web Service and display the Crystal report that is exposed by this service. Web services allow different applications to share both their data and their capabilities. Crystal Reports web services allow your web or windows applications to access reports located on a central report server. Reports stored on a centralized server can be updated independently of the applications that access those reports.

In this tutorial, you learn how to use Crystal Reports web services to populate a DropDownList control with a list of non-embedded reports from a web service.

Adding Controls to the Web or Windows Form

In this section, you add the DropDownList control and Button control to the web or windows form. The instructions for a Web Site are listed first, the instructions for a Windows project are listed second.

> **Note** This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

To create and configure a DropDownList Control and Button Control on the Web Form

1. Open the Web form.
2. From the **View** menu, click **Designer**.
3. Click the **CrystalReportViewer** control to select it.
4. Press the LEFT ARROW on your keyboard so that a flashing cursor appears, and then press ENTER.

   The CrystalReportViewer control drops by one line.
5. From the **Toolbox**, drag a **DropDownList** control above the **CrystalReportViewer** control.

   > **Note** If a Smart Task appears on the DropDownList (when using Visual Studio 2005), press Esc to close it.
6. Click on the **DropDownList** control to select it.
7. From the **Properties** window, set the **ID** to "reportsList"
8. From the **Toolbox**, drag a **Button** control to the right of the **DropDownList** control.
9. Click the **Button** control to select it.

10. From the **Properties** window:

    Set the **ID** to "display"

    Set the **Text** to "Display Report"

11. From the **File** menu, click **Save All**.

To create and configure a DropDownList Control and Button Control on the Windows Form

1. Open the Windows form.
2. From the **View** menu, click **Designer**.
3. From the **Toolbox**, drag a **ComboBox** control to the web form. Place the control above the **CrystalReportViewer** control.

   Note If a Smart Task appears on the **ComboBox** (when using Visual Studio 2005), press Esc to close it.
4. Click on the **ComboBox** control to select it.
5. From the **Properties** window, set the **Name** to "reportsList"
6. From the **Toolbox**, drag a **Button** control to the right of the **ComboBox** control.
7. Click the **Button** control to select it.
8. From the **Properties** window:

   Set the **Name** to "display"

   Set the **Text** to "Display Report"
9. From the **File** menu, click **Save All**.

In the next step, you write code to automatically populate the **DropDownList** or **ComboBox** control.

Adding a Helper Method to Access Crystal Reports Web Services

In this section, you create a helper method that will access a server through Crystal Reports web services and generate a list of reports.

Note This procedure requires access to a machine that has been configured to use Crystal Reports web services. Before proceeding, please ensure that your machine has been configured correctly.

To create a Helper Method to Access Crystal Reports Web Services

1. Open the Web or Windows form.
2. From the **View** menu, click **Code**.
3. Add three private class level variables.

[Visual Basic]

```
Private myServerFileReport As ServerFileReport

Private myReportManagerRequest As ReportManagerRequest

Private myServerFileReportManagerProxy As ServerFileReportManagerProxy
```

[end]

[C#]

```
    private ServerFileReport serverFileReport;
    private ReportManagerRequest reportManagerRequest;
    private ServerFileReportManagerProxy serverFileReportManagerProxy;
```
[end]

4. Create a new private helper method that returns an `ArrayList`.

[Visual Basic]
```
    Protected Function getReports() As ArrayList
    End Function
```
[end]
[C#]
```
    protected ArrayList getReports()
    {
    }
```
[end]

5. Inside of the helper method, instantiate a new `ServerFileReport` instance.

[Visual Basic]
```
    myServerFileReport = New ServerFileReport()
```
[end]
[C#]
```
    serverFileReport = new ServerFileReport();
```
[end]

6. Set the `ReportPath` property of the `ServerFileReport` instance to a null string.

[Visual Basic]
```
    myServerFileReport.ReportPath = ""
```
[end]
[C#]
```
    serverFileReport.ReportPath = "";
```
[end]

7. Set the `WebServiceURL` property of the `ServerFileReport` to the viewer's virtual directory for your installed version of Crystal Reports, see Appendix: Viewers' Virtual Directory.

   In this sample code, the viewers' virtual directory is configured for Crystal Reports for Visual Studio 2005.

[Visual Basic]
```
    myServerFileReport.WebServiceUrl =
    "http://localhost:80/CrystalReportsWebServices2005/serverfilereportserv
    ice.asmx"
```
[end]
[C#]

```
serverFileReport.WebServiceUrl =
"http://localhost:80/CrystalReportsWebServices2005/serverfilereportserv
ice.asmx";
```

[end]

Note This tutorial assumes that your report server is your local machine, thus you use `http://localhost:80/` as the Web Service URL. In order to use a different machine as your report server, replace `http://localhost:80/` with the address and port number of your report server.

8. Instantiate a new `ReportManagerRequest` instance.

[Visual Basic]

```
myReportManagerRequest = New ReportManagerRequest()
```

[end]

[C#]

```
reportManagerRequest = new ReportManagerRequest();
```

[end]

9. Assign the result of the `GetExtraData` method of the `serverFileReport` instance to the `ExtraData` property of the `ReportManagerRequest` instance. This sets the `ExtraData` property to the Web Service URL.

[Visual Basic]

```
myReportManagerRequest.ExtraData = myServerFileReport.GetExtraData()
```

[end]

[C#]

```
reportManagerRequest.ExtraData = serverFileReport.GetExtraData();
```

[end]

10. Assign the result of the `ToUri` method of the `serverFileReport` instance to the `ParentUri` property of the `ReportManagerRequest` instance. This identifies the directory on the server machine that contains the reports to be listed.

[Visual Basic]

```
myReportManagerRequest.ParentUri = myServerFileReport.ToUri()
```

[end]

[C#]

```
reportManagerRequest.ParentUri = serverFileReport.ToUri();
```

[end]

11. Instantiate a new `ServerFileReportManagerProxy` instance.

[Visual Basic]

```
myServerFileReportManagerProxy = New ServerFileReportManagerProxy()
```

[end]

[C#]

```
serverFileReportManagerProxy = new ServerFileReportManagerProxy();
```

[end]

12. Set the `Url` property of the `ServerFileReportManagerProxy` to the Server File Report Manager for your installed version of Crystal Reports, see Appendix: Viewers' Virtual Directory.

    In this sample code, the viewers' virtual directory is configured for Crystal Reports for Visual Studio 2005.

[Visual Basic]

```
myServerFileReportManagerProxy.Url =
"http://localhost:80/CrystalReportsWebServices2005/serverfilereportmanager.asmx"
```

[end]

[C#]

```
serverFileReportManagerProxy.Url =
"http://localhost:80/CrystalReportsWebServices2005/serverfilereportmanager.asmx";
```

[end]

13. Create and instantiate a new `ReportManagerResponse` object.

[Visual Basic]

```
Dim myReportManagerResponse As ReportManagerResponse = new
ReportManagerResponse()
```

[end]

[C#]

```
ReportManagerResponse reportManagerResponse = new
ReportManagerResponse();
```

[end]

14. Call the `ListChildObjects()` method of the `serverFileReportManagerProxy` instance and assign the result to the `ReportManagerResponse` instance. The `ListChildObjects` method returns a list of files located on the report server.

[Visual Basic]

```
myReportManagerResponse =
myServerFileReportManagerProxy.ListChildObjects(myReportManagerRequest)
```

[end]

[C#]

```
reportManagerResponse =
serverFileReportManagerProxy.ListChildObjects(reportManagerRequest);
```

[end]

15. Create and instantiate a new `ArrayList`.

[Visual Basic]

```
Dim myRemoteReports As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList remoteReports = new ArrayList();
```

[end]

16. Next, create a `For Each` loop that loops through each element of `reportManagerResponse`.

[Visual Basic]

```
For Each myReportUriString As String In myReportManagerResponse.ReportUris


Next
```

[end]

[C#]

```
foreach (string reportUriString in reportManagerResponse.ReportUris)
{

}
```

[end]

17. Inside of the `For Each` loop, create a new instance of `serverFileReport`.

[Visual Basic]

```
myServerFileReport = New ServerFileReport()
```

[end]

[C#]

```
serverFileReport = new ServerFileReport();
```

[end]

18. Set the `serverFileReport` Uri to the value of `reportUriString`.

[Visual Basic]

```
myServerFileReport = ServerFileReport.FromUri(myReportUriString)
```

[end]

[C#]

```
serverFileReport = ServerFileReport.FromUri(reportUriString);
```

[end]

19. Still inside of the `For Each` loop, create an `If` conditional block that verifies that the `ObjectType` property of the `ServerFileReport` is of type REPORT.

    Note The `ListChildObjects` method will return both directories and reports. In this tutorial you will filter the list of objects for report files.

[Visual Basic]

```
If (myServerFileReport.ObjectType = EnumServerFileType.REPORT) Then

End If
```

[end]

[C#]

```
if (serverFileReport.ObjectType == EnumServerFileType.REPORT)
{

}
```

[end]

20. Inside of the `If` conditional block, add the `reportUriString` instance to the `remoteReport ArrayList`.

[Visual Basic]

```
myRemoteReports.Add(myReportUriString)
```

[end]

[C#]

```
remoteReports.Add(reportUriString);
```

[end]

21. Finally, outside of the `For Each` loop, return the `remoteReports ArrayList`.

[Visual Basic]

```
Return myRemoteReports
```

[end]

[C#]

```
return remoteReports;
```

[end]

22. From the **File** menu, click **Save All**.

In the next step, you write code to automatically populate the **DropDownList** or **ComboBox** control with a list of reports.

For a Web Site, continue to Populating the DropDownList Control in a Web Site.

For a Windows Project, continue to Populating the ComboBox Control in a Windows Project.

Populating the DropDownList Control in a Web Site

In this section, you write code to populate a DropDownList control with a list of non-embedded reports. You will use the getReports() helper method to generate a list of reports from Crystal Reports web services and then bind this list to the DropDownList control.

For a Windows Project, see Populating the ComboBox Control in a Windows Project.

Populating the DropDownList Control in a Web Site

In this procedure, you populate a sorted list of reports using Crystal Reports Web Services.

1. Open the Web Form.
2. From the **View** menu, click **Code**.
3. Above the class signature, add an `"Imports"[Visual Basic]` or `"using"[C#]` declaration to the top of the class for the System.IO and System.Collections namespaces.

[Visual Basic]

```
Imports System.IO
```

```
Imports System.Collections
```

[end]

[C#]

```
using System.IO;
using System.Collections;
```
[end]

4. Within the `page_load()` event handler, create a `Not IsPostBack` conditional block.

[Visual Basic]
```
If Not IsPostBack Then
Else
End If
```
[end]

[C#]
```
if(!IsPostBack)
{
}
else
{
}
```
[end]

> **Note** The `Not IsPostBack` conditional block is used to encapsulate code that should only be run the first time the page loads. Controls are typically bound to data values within `Not IsPostBack` conditional blocks so that their data values (and any subsequent control events) are not reset during page reloads.

5. Inside of the `If` block, use the `getReports()` helper method to retrieve the list of reports from the Web Service. Assign the result to an ArrayList.

[Visual Basic]
```
Dim myReports As ArrayList = getReports()
```
[end]

[C#]
```
ArrayList reports = getReports();
```
[end]

6. On the next line, instantiate the `SortedList` class and pass it to its parent interface, `IDictionary`.

[Visual Basic]
```
Dim mySortedList As IDictionary = New SortedList
```
[end]

[C#]
```
IDictionary sortedList = new SortedList();
```
[end]

7. Next, create a `For Each` loop that loops through each element of `reports`.

[Visual Basic]
```
For Each myPath As String In myReports
```

```
    Next
```
[end]

[C#]

```
    foreach (string path in reports)
    {
    }
```
[end]

8. The next block of code uses string manipulation methods to remove the file path from the name of each report and replace escape characters with string literals. Insert these lines inside of the `For Each` loop.

[Visual Basic]

```
    Dim myReportNamePrefix As Integer = myPath.LastIndexOf("/") + 1
    Dim myReportNameSufix As Integer = myPath.LastIndexOf("?")
    Dim myReportNameLength As Integer = myReportNameSufix - myReportNamePrefix
    Dim myReportName As String = myPath.Substring(myReportNamePrefix,
    myReportNameLength)
    reportName = reportName.Replace("%20", " ")
```
[end]

[C#]

```
    int reportNamePrefix = path.LastIndexOf(@"/") + 1;
    int reportNameSufix = path.LastIndexOf(@"?");
    int reportNameLength = reportNameSufix - reportNamePrefix;
    string reportName = path.Substring(reportNamePrefix, reportNameLength);
    myReportName = myReportName.Replace("%20", " ");
```
[end]

9. Still inside the `For Each` loop, add a line of code that adds the truncated report name to the `sortedList` IDictionary.

[Visual Basic]

```
    mySortedList.Add(myPath, myReportName)
```
[end]

[C#]

```
    sortedList.Add(path, reportName);
```
[end]

Binding the Sorted List to the DropDownList Control

In this section, you learn how to bind the `IDictionary` data source to the `DropDownList` control. First, you define a relationship between the elements of the `IDictionary` collection and the `DropDownList` fields. Then, you bind the `IDictionary` data source to the `DropDownList` control.

1. Outside of the `For Each` loop, add code to assign the Value field of the `DropDownList` to the key property of the `sortedList` instance.

[Visual Basic]

```
reportsList.DataTextField = "value"
```

[end]

[C#]

```
reportsList.DataTextField = "value";
```

[end]

2. Next, assign the Value field of the `DropDownList` control to the key property of the `sortedList` instance.

[Visual Basic]

```
reportsList.DataValueField = "key"
```

[end]

[C#]

```
reportsList.DataValueField = "key";
```

[end]

3. Assign the `SortedList` instance to the `dataSource` property of the `DropDownList` control.

[Visual Basic]

```
reportsList.DataSource = mysortedList
```

[end]

[C#]

```
reportsList.DataSource = sortedList;
```

[end]

4. Finally, bind the data source to the `DropDownList`.

[Visual Basic]

```
reportsList.DataBind()
```

[end]

[C#]

```
reportsList.DataBind();
```

[end]

You have now completed the contents of the `If Not IsPostBack` conditional block. In the next section, you will write code inside of the `Else` conditional block to define what happens if the page is refreshed without assigning a new value to the `ServerFileReport` instance.

5. Inside of the else statement, assign the ServerFileReport instance to the report that is currently held in session.

[Visual Basic]

```
myServerFileReport = CType(Session("myServerFileReport"),
ServerFileReport)
```

[end]

[C#]

```
serverFileReport = (ServerFileReport)Session["serverFileReport"];
```

[end]

Note Because Session only returns generic objects, you must cast the report to a report type. Whether you use embedded or non-embedded reports, cast the retrieved object to the ServerFileReport type.

6. On the next line, bind the ServerFileReport instance to the CrystalReportViewer control.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = myServerFileReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = serverFileReport;
```

[end]

7. From the **File** menu, select **Save All**.
8. From the **Build** menu, select **Build Solution**.
9. If you have any build errors, go ahead and fix them now.

In the next section, you add a click event to bind the report based on selections from the reportsList DropDownList control.

Populating the ComboBox Control in a Windows Project

In this section, you write code to populate a ComboBox control with a list of non-embedded reports. You will use the getReports() helper method to generate a list of reports from Crystal Reports web services and then bind this list to the ComboBox control.

For a Web Site, see Populating the DropDownList Control.

Populating the ComboBox Control in a Windows Project

In this procedure, you populate a sorted list of reports using Crystal Reports Web Services.

1. Open the Windows Form.
2. From the **View** menu, click **Code**.
3. Above the class signature, add an `"Imports"[Visual Basic]` or `"using"[C#]` declaration to the top of the class for the System.IO, System.Collections and the System.Web.Services namespaces.

[Visual Basic]

```
Imports System.IO
Imports System.Collections
Imports System.Web.Services
```

[end]

[C#]

```
using System.IO;
using System.Collections;
using System.Web.Services
```

[end]

4. Within the `Form1_Load` event handler, use the `getReports()` helper method to retrieve the list of reports from the Web Service. Assign the result to the `reports` array.

[Visual Basic]

```
Dim myReports As ArrayList = getReports()
```

[end]

[C#]

```
ArrayList reports = getReports();
```

[end]

5. On the next line, instantiate an `ArrayList` object.

[Visual Basic]

```
Dim myArrayList As ArrayList = New ArrayList()
```

[end]

[C#]

```
ArrayList arrayList = new ArrayList();
```

[end]

6. Next, create a `For Each` loop that loops through each element of `reports`.

[Visual Basic]

```
For Each myPath As String In myReports
Next
```

[end]

[C#]

```
foreach (string path in reports)
{
}
```

[end]

7. The next block of code uses string manipulation methods to remove the file path from the name of each report and replace escape characters with string literals. Insert these lines inside of the `For Each` loop.

[Visual Basic]

```
Dim myReportNamePrefix As Integer = myPath.LastIndexOf("/") + 1
Dim myReportNameSufix As Integer = myPath.LastIndexOf("?")
Dim myReportNameLength As Integer = myReportNameSufix - myReportNamePrefix
Dim myReportName As String = myPath.Substring(myReportNamePrefix,
myReportNameLength)
myReportName = myReportName.Replace("%20", " ")
```

[end]

[C#]

```
int reportNamePrefix = path.LastIndexOf(@"/") + 1;
int reportNameSufix = path.LastIndexOf(@"?");
```

```
int reportNameLength = reportNameSufix - reportNamePrefix;
string reportName = path.Substring(reportNamePrefix, reportNameLength);
reportName = reportName.Replace("%20", " ");
```

[end]

8. Still inside the `For Each` loop, add a line of code that adds the truncated report name to `arrayList`.

[Visual Basic]

```
myArrayList.Add(myReportName)
```

[end]

[C#]

```
arrayList.Add(reportName);
```

[end]

9. Outside of the `For Each` loop, assign the `ArrayList` instance to the `dataSource` property of the `DropDownList` control. This binds the `ArrayList` data source to the `ComboBox` control.

[Visual Basic]

```
reportsList.DataSource = mysortedList
```

[end]

[C#]

```
reportsList.DataSource = sortedList;
```

[end]

10. From the **File** menu, select **Save All**.
11. From the **Build** menu, select **Build Solution**.
12. If you have any build errors, go ahead and fix them now.

In the next section, you add a click event to bind the report based on selections from the reportsList ComboBox control.

Adding a Click Event to Bind the Report

In this section you configure the display Button control to display the report selected from the reportsList control. Within the event method for this button, you rebind the CrystalReportViewer control to the selected report.

To create the click event method for the Button in a Web project

1. Open the Web or Windows form.
2. From the **View** menu, click **Designer**.
3. Double-click the display Button control.

   You are taken to the code-behind class where a `display_Click()` event method has been automatically generated.

   You are now ready to create the helper method that binds the value of `reportsList` to the `crystalReportViewer` control.
4. Within the `display_Click()` event method that has just been auto-generated, instantiate a new `ServerFileReport`.

[Visual Basic]

```
Dim myServerFileReport As ServerFileReport = New ServerFileReport
```

[end]

[C#]

```
serverFileReport = new ServerFileReport();
```

[end]

5. Set the **ReportPath** property of the `ServerFileReport` instance to the value selected from `reportsList`.

[Visual Basic]

```
myServerFileReport.ReportPath = "\" + reportsList.SelectedItem.ToString
```

[end]

[C#]

```
serverFileReport.reportPath = @"\" + reportsList.selectedItem.toString();
```

[end]

6. Set the **WebServiceUrl** property of the `ServerFileReport` instance to the location of `serverfilereportservice.asmx`.

[Visual Basic]

```
myServerFileReport.WebServiceUrl =
"http://localhost:80/CrystalReportsWebServices2005/serverfilereportserv
ice.asmx"
```

[end]

[C#]

```
serverFileReport.WebServiceUrl =
"http://localhost:80/CrystalReportsWebServices2005/serverfilereportserv
ice.asmx";
```

[end]

7. For a Web Site, assign the report into Session, using the report variable name as the Session identifier string.

[Visual Basic]

```
Session("ServerFileReport") = myServerFileReport
```

[end]

[C#]

```
Session["ServerFileReport"] = serverFileReport;
```

8. Set the ReportSource property of the CrystalReportViewer control to the ServerFileReport instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = myServerFileReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = serverFileReport;
```

[end]

9. From the **File** menu, select **Save All**.

10. From the **Build** menu, select **Build Solution**.
11. If you have any build errors, go ahead and fix them now.

Running the Application

1. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.
2. Select a report from the DropDownList or ComboBox
3. Click **Display Report** to display the report that you selected.

Conclusion

In this tutorial, you learned how to use Crystal Reports web services to populate a DropDownList control with a list of non-embedded reports from a web service.

First, you created a Web or Windows form and added three controls. Then, you populated the DropDownList control or ComboBox with a list of reports generated from Crystal Reports web services. Finally, you created a click event method that binds the report selected from the DropDownList control or ComboBox control to the CrystalReportViewer control.

It is also possible to populate a list of reports from a the file directory of a local machine using Crystal Reports.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in folders that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_WebService

Visual Basic Web Site: VB_Web_WebService

To locate the folders that contain these samples, see Appendix: Tutorials Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Other Tutorial:

Triggered Export with the ReportExporter Control

Triggered Export with the ReportExporter Control

Introduction

In this tutorial, you learn how to use the ReportExporter control to enable triggered exporting when you load a page. To do that, you build a basic Web page with the CrystalReportViewer control, and then replace the CrystalReportViewer control with the ReportExporter control.

In some scenarios, you download exported Crystal reports as part of your daily work routine.

If you do not need to view the report online, the CrystalReportViewer control may not fit your needs. Instead, you might use the ReportExporter control to trigger an export event that downloads the report when you open an ASPX page.

The API of the ReportExporter control is similar to the CrystalReportViewer control, along with the limited object model that is referred to in this documentation as the CrystalReportViewer object model. Most importantly, the ReportSource property is found in both controls, which means that all report binding scenarios are the same for both controls.

So, existing pages and code that currently work with the CrystalReportViewer control can be duplicated, and then reused for triggered export, by replacing the CrystalReportViewer control with the ReportExporter control.

In this tutorial, you learn how to build a basic Web page with the CrystalReportViewer control, and then replace it with the ReportExporter control to enable triggered exporting when you load a page.

Adding the ReportExporter Control to the Toolbox

In this section, you learn how to add the ReportExporter control to the Toolbox in Visual Studio.

First, create a Web Site by following the instructions in Appendix: Project Setup. Then, bind a Crystal report to the CrystalReportViewer control, as described by the instructions in Appendix: Additional Setup Requirements.

To use the ReportExporter control, you must first add it to the Toolbox.

To add the ReportExporter control to the Toolbox

1. From the **Tools** menu, click **Choose Toolbox Items**.

   Note The Choose Toolbox Items dialog box may take a while to open, due to the large number of controls that it accesses.

2. Scroll down the **.NET Framework Components** list, select **ReportExporter**, and then click **OK**.

   The ReportExport control is added to the General node in the Toolbox. You can drag the control to place it under the Crystal Reports node.

Replacing the CrystalReportViewer Control with the ReportExporter Control

In this section, you learn how to replace the CrystalReportViewer control with the ReportExporter control.

To replace the CrystalReportViewer control with the ReportExporter control

Note This procedure works only with a project that has been created from Appendix: Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Appendix: Project Setup.

1. Open the Web Form.
2. From the **View** menu, click **Designer**.
3. Delete the **CrystalReportViewer** control.
4. From the **Toolbox**, drag the **ReportExporter** control onto the Web Form.
5. From the **Properties** window, do the following:
   a) Set the **ID** property to "reportExporter" [C#] or "myReportExporter" [Visual Basic].
   b) Set the **ExportAsAttachment** property to "True."
   c) Set the **ExportFormatType** property to "PDF."
6. From the **View** menu, click **Code**.
7. In the `ConfigureCrystalReports()` method (that you created during Appendix: Project Setup), replace the CrystalReportViewer instance with the ReportExporter instance.

[Visual Basic]

```
myReportExporter.ReportSource = hierarchicalGroupingReport
```

[end]

[C#]

```
reportExporter.ReportSource = hierarchicalGroupingReport;
```

[end]

Testing the ReportExporter Control

In this section, you learn how to test the ReportExporter control to export a Crystal report to one of the available export formats.

To test the ReportExporter control

1. From the **Build** menu, select **Build Solution**.
2. If you have any build errors, go ahead and fix them now.
3. From the **Debug** menu, click **Start**.

   Note If you are developing a Web Site in Visual Studio 2005, and this is the first time you have started debugging, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

   The File Download dialog box is displayed.

4. Click **Save**.
5. Save the report to a location of your choice.
6. In a Web browser, open the saved report.

   The report opens in the export format that you have selected.
7. Return to Visual Studio and click **Stop** to exit from debug mode.

Conclusion

In this tutorial, you learned how to use the ReportExporter control to export a Crystal report to one of the available export formats.

Sample Code Information

Each tutorial comes with Visual Basic and C# sample code that show the completed version of the project. Follow the instructions in this tutorial to create a new project or open the sample code project to work from a completed version.

The sample code is stored in zip files that are categorized by language and project type. The folder names for each sample code version are as follows:

C# Web Site: CS_Web_ReportExporter

Visual Basic Web Site: VB_Web_ReportExporter

To locate the zip files that contain these samples, see Appendix: Tutorials' Sample Code Directory.

Crystal Reports

For Visual Studio 2005

Deployment Tutorials

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Deploying a Windows Application with ClickOnce Deployment

ClickOnce Deployment

Introduction

ClickOnce reduces the time and effort required to deploy Windows applications across a network. Rather than distribute a separate executable to each individual hard drive, ClickOnce places the executable on a common Web page, from which all users can launch it. As part of the launch process, a copy is retrieved to the user's hard drive that can be used to re-launch the application locally. However, this local copy regularly checks the source executable on the Web page for updates.

Updates to the Windows application can be re-published to the Web server, and the newer application files are then available to clients. If an older version of the Windows application opens locally on a client machine, an update dialog box gives the option to check for updates from the Web server.

In this tutorial, you create a new Windows application. Next, you add a CrystalReportViewer control to the Windows Form. Then you create a Crystal report that binds to the control.

Before you publish the Windows application to the Web Site, you set the following properties:

- Security
- Prerequisites
- Automatic Updates

To publish the Windows application, you use the Publishing Wizard to choose a Web Site address. The published Web Site contains hyperlinks to the Windows application installer and the ClickOnce deployment online help. If you have set the prerequisite properties, the Web Site also has a link to the prerequisite files. After the Windows application publishes to the Web Site at the end of this tutorial, you install and run the application on a client machine.

Creating and Binding a Report

In this section, you create a Crystal report that binds to a CrystalReportViewer control.

To set up a Windows project in Crystal Reports for Visual Studio 2005

1. Launch Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Project**.
3. In the **New Project** dialog box, select a language folder for C# or Visual Basic from the **Project Types** list.
4. From the **Templates** list, click **Windows Application**.
5. In the **Name** field, replace the default project name with the name of your project.
6. Click **OK**.

To create and bind a Crystal report to the Windows project

1. Open the Windows Form in **Design** view.
2. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.

3. Drag and drop the **CrystalReportViewer** control onto the form.

   The CrystalReportViewer control supplies a new GUI feature known as Smart Tasks that is titled "CrystalReportViewer Tasks". The Smart Task panel enables you to implement commonly used settings, such as report binding.
4. If the Smart Tasks panel is not open, click the **arrow toggle** on the upper-right corner of the **CrystalReportViewer** control.
5. From the **CrystalReportViewer Tasks** panel, click **Create a new Crystal Report...**.
6. In the **Create a New Crystal Report** dialog box, type "Employees.rpt", and then click **OK**.

   The Crystal report is created and added to the project. A report class is generated with the same name as the report.
7. In the **Create New Crystal Report Document** panel of the **Crystal Reports Gallery** dialog box, select **Using a Report Wizard**.
8. In the **Choose an Expert** panel, select **Standard**, and then click **OK**.
9. In the **Available Data Sources** panel of the **Standard Report Creation Wizard** window, do the following:

   d) Expand the **Create New Connection** folder.

   e) Expand the **ODBC (RDO)** folder.
10. In the **ODBC (RDO)** window, select the correct ODBC DSN entry for your version of Crystal Reports as explained in Appendix: ODBC DSN Entry for Xtreme Sample Database, and then click **Finish**.

    The ODBC (RDO) folder expands and shows the Xtreme Sample Database.
11. Expand the **Tables** node, double-click the **Employee** table to move the table to the **Selected Tables** panel, and then click **Next**.
12. Expand the **Employee** table, then CTRL-click **Employee ID**, **Last Name**, **First Name**, and **Salary**.
13. Click the **>** symbol to move these fields into the **Fields to Display** panel, then click **Next**.
14. In the **Available Fields** panel, under **Report Fields**, double-click **Employee.Employee ID** to move the field into the **Group By** panel, and then click **Finish**.

    The Employees report is created and loaded into the main window of Visual Studio.

    At the bottom of the report designer, click **Main Report Preview** to preview the report with dummy data.

To test the loading of the CustomersByCity report

1. Return to the Windows Form in **Design** view.

   The Employees report is displayed as "Employees1" in the Component tray at the bottom of the window. The CrystalReportViewer control displays the Employees report.
2. From the **Build** menu, select **Build Solution**.
3. If you have any build errors, go ahead and fix them now.
4. From the **Debug** menu, click **Start**.

The Employees report is displayed successfully with the dummy data.

5. Return to Visual Studio and click **Stop** to exit out of debug mode.

Setting the ClickOnce Deployment Properties

Before you deploy the Windows application, you set properties for Security and Publish from the application's Properties window.

The Security property sets the permissions for your application. By default, the ClickOnce deployment has a limited subset of permissions. You can set the permissions to deploy only to client machines on the Intranet or Internet, or to allow full permissions.

The Publish property sets the deployment location to a Web Site URL, an FTP server, or a file path. Also, this property sets necessary prerequisites and updates to the deployed application.

To set the ClickOnce Deployment Options

1. In the **Solution Explorer**, right-click the bold project name, and click **Properties**.
2. Click on the **Security** tab.
3. Select **Enable ClickOnce Security Settings**.

   By default, the ClickOnce security permission is set to **This is a full trust application**.
4. Click on the **Publish** tab, and then click **Prerequisites**.
5. In the **Prerequisites** dialog box, under the **Choose which prerequisites to install** list, select **.NET Framework 2.0** and **Crystal Reports for .NET Framework 2.0.**

   Note Encourage your clients to install the .NET Framework and Crystal Reports .NET Framework 2.0 runtime before you deploy the Windows application, because installation time increases significantly if you use ClickOnce to install the prerequisites.
6. Click **OK**.
7. On the **Publish** tab, click **Updates**. From the **Application Updates** dialog box, ensure the following options are selected:
   a) Select **The application should check for updates**.
   b) In the **Choose when the application should check for updates** list, select **Before the application starts**.
8. Click **OK** to close the dialog box.
9. From the **File** menu, click **Save All**.

Publishing the Windows Application

In this section, you learn to use the Publishing Wizard to publish the Windows application to a Web Site.

To publish a Windows application

1. In the **Solution Explorer**, right-click the bold project name, and click **Publish**.
2. When the **Publish Wizard** dialog box appears, click **Next**.
3. In the **Will the application be available offline?** dialog box, select **Yes, this application is available online or offline**, and then click **Next**.

4. From the **Ready to Publish!** dialog box, note the URL where the application is published, and then click **Finish**.

Installing the Windows Application on a Client Machine

To complete ClickOnce deployment, open the published Web Site and choose the option to install the Windows application.

Note Encourage your clients to install the .NET Framework and Crystal Reports .NET Framework 2.0 runtime before you deploy the Windows application, because installation time increases significantly if you use ClickOnce to install the prerequisites.

To install and run the Windows application on a client machine

1. On the client machine, open a Web browser window.
2. In the browser's address bar, type the URL of the published Web Site that you have created in the previous procedure.

   The published Web Site displays the name of your Windows application.
3. The Web Site contains links to install and run the Windows application with the prerequisites or without the prerequisites.
   a) Click the **launch** link to run the application if you have already installed the prerequisites on the client machine.
   b) Click the **install** link to install the prerequisites on the client machine and run the application.

      From the **Install Application** dialog box, the **Grant permissions and confirm installation** message appears.

      Note This message appears because you set the security property to FullTrust in the procedure Setting the ClickOnce Deployment Properties.
4. Click **Install**.

   After the installation is complete, you can access the Windows application through the Start menu.

Conclusion

You have successfully created, published and deployed a Windows application that uses Crystal Reports.

When you modify the application on the development machine, re-publish the application and set the update properties. If an older version of the application opens on the client machine, the client is notified about a new version of the Windows application.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Creating a New Web Site Deployment Project with Windows Installer

Creating a new Web Site Deployment Project with Windows Installer

To deploy your Web Site, follow the tutorial instructions in this section.

First, you create a Web Setup Project to deploy a Web Site that uses Crystal Reports for Visual Studio 2005. Then, add output files that are necessary for the application to run. Finally, build the installer files that will deploy your Web Site.

Creating a Web Setup Project

In this section, you create a Web Setup Project from the deployment projects that are available in Visual Studio. You must have a completed Web Site that uses Crystal Reports. For an example, see the Appendix: Project Setup.

To create a Web Setup project for Web Sites

1. In Visual Studio, open your Web Site.
2. On the **File** menu, point to **Add**, and then click **New Project**.
3. In the **Project Types** panel of the **Add New Project** dialog box, select **Other Project Types** and click **Setup and Deployment Projects**.
4. In the **Templates** panel, select **Web Setup Project**.
5. Choose an appropriate name for the project and specify its location and then click **OK**.

   For the purposes of this tutorial, the setup project is referred to by the default name "WebSetup1".
6. On the **File System** tab, expand **Web Application Folder**.
7. In the **Properties** window, set the **DefaultDocument** property to the start page (an ASPX file) for the Web Site.

   The **VirtualDirectory** property of the Web Setup Project has been set to the project name.

Adding Components to the Web Setup Project

Since you plan to use the .msi Windows Installer file to deploy the Web Site, you do not need to add merge modules to the setup project. See Appendix: Crystal Reports for .NET Framework 2.0 Windows Installer for more information.

To add output files to the Web Setup Project

1. In **Solution Explorer**, right-click **WebSetup1**, point to **Add**, and then click **Project Output...**.
2. In the **Add Project Output Group** dialog box, select **Content Files**.

   Leave the **Configuration** as (Active).

Building and Deploying the Web Setup Project

Building the Web Setup Project creates the installer files to copy to other computers. You can run either of these installers on the target computer to deploy the Web Site.

Note Always ensure that the target computer already has the .NET Framework installed. For more information about the .NET Framework, see Appendix: NET Framework 2.0.

To build the Web Setup Project

1. Select **WebSetup1** in **Solution Explorer**.
2. From the **Build** menu, click **Build WebSetup1**.

   The build process creates the following installer files: setup.exe and WebSetup1.msi.

To deploy the Web Setup Project

1. Outside of Visual Studio, navigate to the directory where your deployment project has been saved.
2. Double-click the **WebSetup1** folder.
3. Open the **Debug** folder to find the files that were built by the Web Setup project.
4. Copy the .msi Windows Installer file to the target computer.
5. Distribute the Crystal reports that are used in the Web Site.
6. On the target computer, double-click **WebSetup1.msi** to install the Web Site with Windows Installer.
7. To view the deployed Web Site, open a Web browser window on the target computer. Then type "http://localhost/WebSetup1" in the address bar.

   Replace "localhost" with the name of your server.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Creating a New Windows Application Deployment Project with Windows Installer

Creating a new Windows Application Deployment Project with Windows Installer

To deploy your Windows applications, follow the tutorial instructions in this section.

First, you create a Setup Project for Windows Applications to deploy a Windows application that uses Crystal Reports for Visual Studio 2005. Then, add output files that are necessary for the application to run. Finally, you build the installer files that will deploy your Windows application.

Creating a Setup Project for Windows Applications

In this section, you create a Setup Project for Windows Applications from the deployment projects that are available in Visual Studio. You need to have a completed Windows application that uses Crystal Reports. For an example, see the Appendix: Project Setup.

To create a Setup project for Windows applications

1. In Visual Studio, open your Windows application.
2. On the **File** menu, point to **Add** and then click **New Project**.
3. In the **Project Types** panel of the **Add New Project** dialog box, select **Other Project Types** and click **Setup and Deployment Projects**.
4. In the **Templates** panel, select **Setup Project**.
5. Choose an appropriate name for the project and specify its location, and then click **OK**.

   For the purposes of this tutorial, the setup project is referred to by the default name "Setup1".

Adding Components to the Setup Project

Since you plan to use the .msi Windows Installer file to deploy the Windows application, you do not need to add merge modules to the setup project. See Appendix: Crystal Reports for .NET Framework 2.0 Windows Installer for more information.

To add the Windows application's project output

1. In **Solution Explorer**, right-click **Setup1**, point to **Add**, and then click **Project Output...**.
2. In the **Add Project Output Group** dialog box, select **Primary output**.

   Leave the **Configuration** as (Active).

Building and Deploying the Setup Project

Building the Setup Project creates the installer files to copy to other computers. You can run either of these installers on the target computer to deploy the Windows application.

> **Note** Always ensure that the target computer already has the .NET Framework installed. For more information about the .NET Framework, see Appendix: NET Framework 2.0.

To build the Setup Project

1. In **Solution Explorer**, select **Setup1**.

2. From the **Build** menu, choose **Build Setup1**.

   The build process creates the following installer files: setup.exe and Setup1.msi.

To deploy the Setup Project

1. Outside of Visual Studio, navigate to the directory where your deployment project has been saved.
2. Double-click the **Setup1** folder.
3. Open the **Debug** folder to find the files that were built from the Setup Project.
4. Copy the .msi Windows Installer file to the target computer.
5. Distribute the Crystal reports that are used in the Windows application.
6. On the target computer, double-click **Setup1.msi** to install the Windows application with Windows Installer.
7. To view the Windows application, navigate to its installed location, and then double-click the Windows application .exe file.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Migrating a Project that Uses Crystal Reports for Visual Studio .NET 2002 or 2003 Merge Modules Deployment

Migrating a Project that Uses Crystal Reports for Visual Studio .NET 2002 or 2003 Merge Modules Deployment

To migrate a project that uses Crystal Reports for Visual Studio .NET 2002 or 2003 from Merge Modules deployment to Crystal Reports for Visual Studio 2005 Windows Installer deployment, follow the instructions in this section.

First, convert the Windows application or Web Site that uses Crystal Reports for Visual Studio .NET 2002 or 2003 to a Visual Studio 2005 project. Then, create a new setup project to deploy the Windows application or Web Site. Finally, use the .msi Windows Installer file to deploy your application.

To convert a Visual Studio .NET 2002 or 2003 project to a Visual Studio 2005 project

1. In Visual Studio, on the **File** menu, point to **Open** and then click **Project/Solution...**.
2. Select the Visual Studio .NET 2002 or 2003 Windows application or Web Site, and click **Open**.

   The **Visual Studio Conversion Wizard** opens.
3. Follow the steps **Visual Studio Conversion Wizard** and click **Finish** to convert the project to Visual Studio 2005.

To create a new setup project

1. In **Solution Explorer**, right-click an existing setup project, and click **Remove**.

   Repeat this step for any other existing setup projects.
2. Create a new setup project:

   For a Web Site, see Creating a new Web Site Deployment Project with Windows Installer.

   For a Windows Application, Creating a new Windows Application Deployment Project with Windows Installer.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Migrating a Project that Uses Crystal Reports for Visual Studio 2005 Merge Modules Deployment

Migrating a Project that Uses Crystal Reports for Visual Studio 2005 Merge Modules Deployment

To migrate an existing project that uses Crystal Reports for Visual Studio 2005 from Merge Modules deployment to Windows Installer deployment, follow the instructions in this section.

First, remove the merge modules from the setup project. Then, re-build the setup project. Finally, use the .msi Windows Installer file to deploy your application.

For the purposes of this tutorial, the setup project is referred to by the default name "Setup1".

To remove merge modules from the setup project

1. In Visual Studio, open your Windows application or Web Site.
2. In **Solution Explorer**, expand **Setup1**, point to a merge module that has been added to the setup project, and then click **Remove**.
3. Repeat Step 2 to remove all Crystal Reports for Visual Studio 2005 merge modules.

To build the Setup Project

1. Select **Setup1** in **Solution Explorer**.
2. From the **Build** menu, choose **Rebuild Setup1**.

   The build process creates the following installer files: setup.exe and Setup1.msi.

To deploy the Setup Project

1. Outside of Visual Studio, navigate to the directory where your deployment project has been saved.
2. Double-click the **Setup1** folder.
3. Open the **Debug** folder to find the files that were built from the Setup Project.
4. Copy the .msi Windows Installer file to the target computer.
5. Distribute the Crystal reports that are used in the project.
6. On the target computer, double-click **Setup1.msi** to install the Windows application or Web Site with Windows Installer.
7. Complete the setup.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Performing a Silent Installation with a Windows Installer

Performing a Silent Installation with a Windows Installer

With a Windows Installer you can perform a silent installation on a client machine. A silent installation is an installation that is done entirely from the command line, and requires no interaction.

To Perform a Silent Installation with a Windows Installer

1. Go to the command prompt.
2. Locate the path to the Windows Installer file. This tutorial assumes that the Windows Installer is located at "`C:\install.msi`".
3. Enter the below command. Use a valid product key code as the value for `PIDKEY`.

   ```
   msiexec.exe /i "C:\install.msi" /qn PIDKEY=AAAAA-1111111-1111111-BBBB
   ```

   The application installs.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Deploying Web Sites with Merge Modules

Deploying Web Sites With Merge Modules

To deploy your Web Site, follow the tutorial instructions in this section.

First, you create a Web Setup Project to deploy a Web Site that uses Crystal Reports for Visual Studio 2005. Then, add output files and merge modules necessary for the application to run. Finally, build the installer files that will deploy your Web Site.

Creating a Web Setup Project

In this section, you create a Web Setup Project from the deployment projects that are available in Visual Studio. You must have a completed Web Site that uses Crystal Reports. For an example, see the Appendix: Project Setup.

To create a Web Setup project for Web Sites

1. In Visual Studio, open your Web Site.
2. On the **File** menu, point to **Add**, and then click **New Project**.
3. In the **Project Types** panel of the **Add New Project** dialog box, select **Other Project Types** and click **Setup and Deployment Projects**.
4. In the **Templates** panel, select **Web Setup Project**.
5. Choose an appropriate name for the project and specify its location and then click **OK**.

   For the purposes of this tutorial, the setup project is referred to by the default name "WebSetup1".
6. On the **File System** tab, expand **Web Application Folder**.
7. In the **Properties** window, set the **DefaultDocument** property to the start page (an ASPX file) for the Web Site.

   The **VirtualDirectory** property of the Web Setup Project has been set to the project name.

Adding Components to the Web Setup Project

This section adds the merge modules and output files used by the Web Site to the Web Setup project. However, if you plan to use the .msi Windows Installer file to deploy the Web Site, you do not need to add merge modules to the setup project. See Appendix: Crystal Reports for .NET Framework 2.0 Windows Installer.

To add output files to the Web Setup Project

1. In **Solution Explorer**, right-click **WebSetup1**, point to **Add**, and then click **Project Output...**.
2. In the **Add Project Output Group** dialog box, select **Content Files**.

   Leave the **Configuration** as (Active).

   If you are using the Windows Installer on the target machine, skip the next procedure and continue to Building and Deploying the Web Setup Project.

To add merge modules to the Web Setup Project

1. In **Solution Explorer**, right-click **WebSetup1**, point to **Add**, and then click **Merge Module...**.
2. In the **Add Modules** dialog box, select the merge modules that are required for your Web Site.

For a description of which merge modules to use for different machines, see Appendix: Crystal Reports Merge Modules for Visual Studio 2005.

3. In the **Solution Explorer**, ensure that your selected merge modules were added under **WebSetup1**.

Building and Deploying the Web Setup Project

Building the Web Setup Project creates the installer files to copy to other computers. You can run either of these installers on the target computer to deploy the Web Site.

Note Always ensure that the target computer already has the .NET Framework installed. For more information about the .NET Framework, see Appendix: NET Framework 2.0.

To build the Web Setup Project

1. In **Solution Explorer**, select **WebSetup1**.
2. From the **Build** menu, click **Build WebSetup1**.

   The build process creates the following installer files: setup.exe and WebSetup1.msi.

To deploy the Web Setup Project

1. Outside of Visual Studio, navigate to the directory where your deployment project has been saved.
2. Double-click the **WebSetup1** folder.
3. Open the **Debug** folder to find the files that were built by the Web Setup project.
4. Copy all the files to the target computer.
5. Distribute the Crystal reports that are used in the Web Site.
6. On the target computer, double-click **Setup.exe** or **WebSetup1.msi** to install the Web Site.
7. To view the deployed Web Site, open a Web browser window on the target computer. Then type "http://localhost/WebSetup1" in the address bar.

   Replace "localhost" with the name of your server.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Deploying Windows Applications with Merge Modules

Deploying Windows Applications

To deploy your Windows applications, follow the tutorial instructions in this section.

First, you create a Setup Project for Windows Applications to deploy a Windows application that uses Crystal Reports for Visual Studio 2005. Then, add output files and merge modules necessary for the application to run. Finally, you build the installer files that will deploy your Windows application.

Creating a Setup Project for Windows Applications

In this section, you create a Setup Project for Windows Applications from the deployment projects that are available in Visual Studio. You need to have a completed Windows application that uses Crystal Reports. For an example, see the Appendix: Project Setup.

To create a Setup project for Windows applications

1. In Visual Studio, open your Windows application.
2. On the **File** menu, point to **Add** and then click **New Project**.
3. In the **Project Types** panel of the **Add New Project** dialog box, select **Other Project Types** and click **Setup and Deployment Projects**.
4. In the **Templates** panel, select **Setup Project**.
5. Choose an appropriate name for the project and specify its location, and then click **OK**.

   For the purposes of this tutorial, the setup project is referred to by the default name "Setup1".

Adding Components to the Setup Project

Add the merge modules and output files used by the Windows application to the setup project. However, if you plan to use the .msi Windows Installer file to deploy the Windows application, you do not need to add merge modules to the setup project. See Appendix: Crystal Reports for .NET Framework 2.0 Windows Installer for more information.

To add the Windows application's project output

1. In **Solution Explorer**, right-click **Setup1**, point to **Add**, and then click **Project Output...**.
2. In the **Add Project Output Group** dialog box, select **Primary output**.

   Leave the **Configuration** as (Active).

   If you are using the Windows Installer on the target machine, skip the next procedure and continue to Building and Deploying the Setup Project.

To add merge modules to the Windows Setup Project

1. In **Solution Explorer**, right-click **Setup1**, point to **Add**, and then click **Merge Module...**.
2. In the **Add Modules** dialog box, select the merge modules that are required for your Windows application.

   For a description of which merge modules to use for different machines, see Appendix: Crystal Reports Merge Modules for Visual Studio 2005.

3. In the **Solution Explorer**, ensure that your selected merge modules were added under **Setup1**.

Building and Deploying the Setup Project

Building the Setup Project creates the installer files to copy to other computers. You can run either of these installers on the target computer to deploy the Windows application.

Note Always ensure that the target computer already has the .NET Framework installed. For more information about the .NET Framework, see Appendix: NET Framework 2.0.

To build the Setup Project

1. In **Solution Explorer**, select **Setup1**.
2. From the **Build** menu, choose **Build Setup1**.

   The build process creates the following installer files: setup.exe and Setup1.msi.

To deploy the Setup Project

1. Outside of Visual Studio, navigate to the directory where your deployment project has been saved.
2. Double-click the **Setup1** folder.
3. Open the **Debug** folder to find the files that were built from the Setup Project.
4. Copy all the files to the target computer.
5. Distribute the Crystal reports that are used in the Windows application.
6. On the target computer, double-click **Setup.exe** or **Setup1.msi** to install the Windows application.
7. To view the Windows application, navigate to its installed location, and then double-click the Windows application .exe file.

Crystal Reports

For Visual Studio 2005

Deployment Tutorial:

Configuring Database Driver Installation Options

Configuring Database Driver Installation Options

In Crystal Reports for Visual Studio 2005, the merge module that you have added to your deployment project allows you to include or exclude specific Crystal Reports database drivers.

To change an option for a specific database driver

1. In Visual Studio, open your deployment project.
2. In **Solution Explorer**, expand the node of the setup project.
3. Select the merge module that you have added to your deployment project based on the deployment machine's type. Check its name from the list of available merge modules in Appendix: Crystal Reports Merge Modules for Visual Studio 2005.
4. In the **Properties** window, expand the **MergeModulesProperties** node.

   A table with the database drivers that are available for installation appears.
5. Set the value of a specific driver:

   Set the value to **1** to include the driver in the deployment project.

   Set the value to **0** to exclude the driver from the deployment project.

   Note By default, all database drivers are included to install with the merge module.

Property name for database driver	Description
InstallCRDB_ADO	Crystal Reports database driver for Microsoft ActiveX Data Objects/OLE DB
InstallCRDB_ADOPLUS	Crystal Reports database driver for Microsoft ADO.NET
InstallCRDB_DAO	Crystal Reports database driver for Microsoft Data Access Objects
InstallCRDB_DATASET	Crystal Reports database driver for DataSet provider
InstallCRDB_ODBC	Crystal Reports database driver for ODBC

Crystal Reports

For Visual Studio 2005

Appendix

Crystal Reports Product Keycode and Registration Number

The product keycode is not the registration number. There are three primary differences:

Format

Product Keycode - Alphanumeric

The product keycode is an alphanumeric string that has a length of 19 digits or characters.

Registration Number - Numeric

The registration number has a length of 10 digits.

Location

Product Keycode - Help>About Microsoft Visual Studio Menu

To locate the product keycode, go to the Help menu, click About Microsoft Visual Studio, and locate Crystal Reports in the Installed Products list. The product keycode is listed next to the Crystal Reports product name.

Registration Number - via Email

The registration number is emailed to you *after* you register your product using the Registration Wizard and the product keycode.

Purpose

Product Keycode

You will need the product keycode to deploy Web or Windows applications that use the Crystal Reports for Visual Studio 2005 .NET SDK merge modules or Windows Installer.

Registration Number

Registration assures that you will be notified whenever the product is upgraded to offer new features, benefits, and efficiencies.

Using the Product Keycode for Deployment Projects

In Crystal Reports for Visual Studio 2005, when you create deployment projects that use the Windows Installer or merge modules for certain report binding scenarios, you need a product keycode to validate the licensing for Web or Windows applications.

To locate the Product Keycode in Visual Studio

1. In Visual Studio, on the **Help** menu, select **About Microsoft Development Environment**.
2. On the Installed Products list, the product keycode is found next to the line that specifies the Crystal Reports version for Visual Studio.

The Crystal Reports Registration Number

When you create reports using Crystal Reports for Visual Studio 2005, the Business Objects Registration Wizard appears.

Registration assures that you will be notified whenever the product is upgraded to offer new features, benefits, and efficiencies.

After you register Crystal Reports for Visual Studio 2005 using your product keycode, you will receive a 10 digit registration number through email.

Note The Business Objects Registration Wizard dialog box does not appear when you have already registered your version of Crystal Reports.

To register Crystal Reports, you need to follow the instructions on the Business Objects Registration Wizard dialog box.

To register in Visual Studio

1. In Visual Studio, open a Crystal report.
2. From the Crystal Reports menu, select **Register / Change Address...**.
3. Follow the instructions on the Business Objects Registration Wizard dialog box.

Design Time Preview

Design time preview is the ability to preview the report at design time. In previous versions of Crystal Reports for Visual Studio .NET, the CrystalReportViewer control in Web projects offered design time preview, when the report was assigned as a file directory path to the ReportSource property in the Properties window.

This feature has now been added to Windows projects. When the report is assigned to the ReportSource property in the Properties window, a preview of the report is displayed in the Windows Form at design time.

Formula Reference

When creating formulas, you have the option of using either Crystal or Basic syntax. Almost any formula written with one syntax can be written with the other. Reports can contain formulas that use Basic syntax as well as formulas that use Crystal syntax, but a single formula can use only one syntax.

If you are familiar with Microsoft Visual Basic or other versions of Basic, then Basic syntax may be more familiar to you. In general, Basic syntax is similar to Visual Basic except that it has specific extensions to handle reporting.

For descriptions and examples of individual functions and operators, go to http://support.businessobjects.com/search/ and search for *cr8_formularef.zip*.

Basic Syntax

When creating formulas, you have the option of using either Crystal or Basic syntax. If you are familiar with Microsoft Visual Basic or other versions of Basic, then Basic syntax may be more familiar to you. In general, Basic syntax is similar to Visual Basic except that it has specific extensions to handle reporting.

Using Basic syntax does not slow report processing down. Reports using Basic syntax formulas can run on any machine that Crystal Reports runs on. Also, using Basic syntax formulas does not require distributing any additional files with your reports.

For descriptions and examples of individual functions and operators, go to http://support.businessobjects.com/search/ and search for *cr8_formularef.zip*.

Basic Syntax Fundamentals

Familiar Features

Many Basic syntax functions work in the same way as their counterparts in Visual Basic. This includes string functions such as Len, Mid and Filter, math functions such as Abs, Rnd and Sin, financial functions such as PV, programming shortcut functions such as IIF, and date functions such as DateSerial, DateAdd and DateDiff.

Most operators supported by Visual Basic are also in Basic syntax. For example, string concatenation (&) and date-time literals (#...#).

Most statements and control structures use the same syntax as in Visual Basic. This includes the If, Select, Do While, Do Until, While and For/Next statements.

Basic style comments and line continuation characters are supported, as is the Basic language use of new lines, colons, and the equal sign.

The Result of a Formula

The result of a formula, or the value that is printed when the formula is placed in a report, is called the value returned by the formula. Every formula in Crystal Reports must return a value. Basic syntax does this by setting the value of the special variable "formula". For example, here is a simple Basic syntax formula that returns the value 10:

```
formula = 10
```

The value returned by a formula can be one of the seven simple data types supported: Number, Currency, String, Boolean, Date, Time and DateTime. Crystal Reports also supports range types and array types, but these cannot be returned by a formula.

The variable that is named formula must be assigned a value

If the variable that is named formula is not assigned a value, it is not a complete Basic syntax formula.

Sometimes you may want to write a formula that just declares and initializes some global variables. These formulas are commonly inserted into the report header section of a report. In such cases, assign any value to the special variable that is named formula. Every formula must return a value, even if you are not interested in using that value.

Example

```
Rem Some Global variable declarations
Rem Remember to set the value of 'formula'
Global x As String, y As Number, z As DateTime
x = "hello"
y = 10.5
z = #Aug 6, 1976#
formula = 10
```

Data Types and the Variable that is named Formula

The variable that is named formula can be set several times within a single formula. For example, suppose a company has a shipping policy in which orders over $1,000 are insured, but orders below that amount are not insured:

```
Rem A formula that returns a String value
If {Orders.Order Amount} >= 1000 Then
   formula = "Insured shipping"
Else
   formula = "Regular shipping"
End If
```

The above variable that is named formula returns the text string value "Insured shipping" if the value of the database field {Orders.Order Amount} is greater than or equal to 1000; otherwise, it returns the text string value "Regular Shipping." Notice that this variable appears twice in the above example.

If the variable that is named formula is set to a value of one type, it cannot be set to a value of another type later in the same formula. For example, replacing the String "Regular Shipping" in the above example with the Number 10 would result in an error since the special variable formula was first set to the String value "Insured shipping."

The reason for this restriction is that Crystal Reports needs to know in advance what the return type of a formula will be so that it can allocate enough storage for the returned values. This is because different types have different storage requirements. Another reason is that the formatting options available for a formula field depend on its type. For

example, a Number field has Number formatting options, such as the number of decimals to display, which do not make sense for a String field.

Note The special variable that is named formula should not be declared, unlike other variables used in a Basic syntax formula.

Case Sensitivity

Basic syntax is not case-sensitive. What this means is that "formula", "Formula" and "FORMULA" are all considered to be the same. This is true of all variable names, functions, and keywords used in a Basic syntax formula.

Note The only exception to this rule is for strings. The string "Hello" is not the same as the string "hello".

Practice using the Xtreme Sample Database

Many of the examples in this section refer to the Xtreme sample database.

Note For information on configuring this database and its ODBC entry, see Appendix: Location of Xtreme Sample Database and ODBC DSN Entry for Xtreme Sample Database.

Comparing Basic Syntax Formulas with Functions in Visual Basic

The use of the *formula* variable is similar to writing a function named *formula* in Visual Basic.

Consider the following Basic syntax formula:

```
Rem A formula that returns a String value
Rem The function Rnd returns a random number
Rem between 0 and 1
If Rnd > 0.9 Then
   formula = "You won!"
Else
   formula = "Sorry, try again."
End If
```

The above formula returns the text string value "You won!" if the random number returned by Rnd is greater than 0.9 and the text string value "Sorry, try again." otherwise.

The above formula could be written as a Visual Basic function as follows:

```
Rem The following code is in Visual Basic
Function formula()
    If Rnd > 0.9 Then
       formula = "You won!"
    Else
       formula = "Sorry, try again."
```

```
    End If
End Function
```

Comments (Basic Syntax)

Formula comments are notes included with a formula to explain its design and operation. Comments do not print and they do not affect the formula; they appear only in the Formula Editor. Use comments to explain the purpose of a formula or explain the steps involved in writing it.

Comments work as in Visual Basic. Begin comments with a *Rem* or an apostrophe.

> **Note** A comment beginning with *Rem* is a separate statement and must either start on a new line or be separated from the previous statement by a colon.

```
Rem This is a comment
Rem This is another comment
formula = 10 'So is any text after an apostrophe
formula = 20 : Rem This is also a comment
'Comments can occur after the formula text
```

Fields (Basic Syntax)

Many of the fields you use when creating your report can also be referred to in your formulas. For example, database, parameter, running total, SQL expression, summary, and group name fields can all be used in a formula. You can also refer to other formula fields in your formula.

The easiest way to insert a field into your report is to double-click a field's name in the Report Fields tree. This ensures that the correct syntax for the field is used.

How Fields Appear in Formulas

Database, parameter, formula, running total and SQL expression fields have their names surrounded by braces.

Database field names are taken from the database: `{Employee.Last Name}`

Parameter, formula, running total, and SQL expression field names are specified when the fields are created.

- parameter fields also includes a question mark: `{?my parameter field}`
- formula fields include an at sign: `{@another formula }`
- running totals fields include a pound sign: `{#my running total}`
- SQL expression fields include a percent sign: `{%my SQL expression}`

Summary and group name fields look like function calls. However, they are really shorthand notation for a report field.

- sum summary field: `Sum({Orders.Order Amount}, {Orders.Ship Via})`
- group name field: `GroupName({Orders.Ship Via})`

Example

The formula in this example uses the Xtreme database. To find out how many days it takes to ship the product from the date when the order was placed, subtract the ship date database field from the order date database field:

```
Rem A formula that uses database fields
formula = {Orders.Ship Date} - {Orders.Order Date}
```

To find the total dollar amount of a given product that was ordered, multiply its unit price by the quantity ordered:

```
formula = {Orders Detail.Unit Price} * _
          {Orders Detail.Quantity}
```

Note The example uses the line continuation character " _ " (space underscore).

To calculate a sale price of 80 percent of the original unit price:

```
formula = {Orders Detail.Unit Price} * 0.80
```

Statements (Basic Syntax)

A Basic syntax formula consists of a sequence of statements. Each statement must be separated from the previous statement by either a new line or a colon. Typically, each statement takes one line, but you can continue a statement onto the next line by using the line continuation character, which is a space followed by an underscore.

Example

```
'Declare a variable x to hold a number
Dim x As Number
'Assign the value of 30 to x
x = 10 + 10 + 10
'This also assigns the value of 30 to x
x = 10 + _
    10 + 10
'Line continuation characters _
 can also be used in comments
Dim y as String
'Three statements separated by two colons
y = "Hello" : x = 30 : formula = True
```

Assignment (Basic Syntax)

Use the equal sign (=) when making assignments. The keyword *Let* can be optionally included as well.

Example

```
x = 10
Let y = 20
```

Simple Data Types (Basic Syntax)

The simple data types in Crystal Reports are:

- Number (Basic Syntax)
- Currency (Basic Syntax)
- String (Basic Syntax)
- Boolean (Basic Syntax)
- Date, Time and DateTime (Basic Syntax)

Number (Basic Syntax)

Enter numbers without any comma separators or currency symbols. (Generally, you would want to have formatted numbers appearing as the result of a formula and not in the formula itself.)

Examples

```
10000
-20
1.23
```

Currency (Basic Syntax)

Use the CCur function to create a Currency value. The initial C in CCur stands for convert and it can be used to convert Number values to Currency values.

Examples

```
CCur (10000)
CCur (-20)
CCur (1.23)
```

String (Basic Syntax)

Strings are used to hold text. The text must be placed between double quotation marks (") and cannot be split between lines. If you want to include double quotes in a string, use two consecutive double quotation marks.

```
"This is a string."
"123"
"The word ""hello"" is quoted."
```

You can extract individual elements or substrings from a string by specifying the character position or a range of character positions. Negative values are allowed; they specify the position starting from the end of the string.

```
"hello" (2) 'Equal to "e"
"hello" (-5) 'Equal to "h"
"604-555-1234" (1 to 3) 'Equal to "604"
"abcdef" (-3 to -1) 'Equal to "def"
```

You can also extract substrings from a string using the Left, Right and Mid functions.

Boolean (Basic Syntax)

The Boolean values are:

```
True
```

```
False
```

Note *Yes* can be used instead of True and *No* instead of False.

Date, Time, and DateTime (Basic Syntax)

The DateTime type can hold date-times, dates only or times only. The Date type holds dates only and the Time type holds times only. The Date and Time types are more efficient than the DateTime type, and so can be used in situations where the added functionality and flexibility of the DateTime type is not needed.

Visual Basic does not support separate types for holding dates only or times only. The Basic syntax DateTime type is similar to Visual Basic's Date type.

You can create DateTime values directly using the date-time literal construction. It is formed by typing in the date-time between two pound (#) signs. Many different formats are supported, as in Visual Basic.

Note Date-time literals cannot be split between lines.

Examples

```
#8/6/1976 1:20 am#
```

```
#August 6, 1976#
```

```
#6 Aug 1976 13:20:19#
```

```
#6 Aug 1976 1:30:15 pm#
```

```
#8/6/1976#
```

```
#10:20 am#
```

Even though #10:20 am# looks like it could have the Time type and #8/6/1976# looks like it could have the Date type, they do not. They both have the DateTime type, as do all date-time literals. For example, you can think of #10:20 am# as a DateTime value with a null date part. To convert it to the Time type use CTime (#10:20 am#).

Instead of using date-time literals you can use CDateTime to convert a String to a DateTime. For example,

```
CDateTime ("8/6/1976 1:20 am")
```

```
CDateTime ("10:20 am")
```

However, there is one key difference between using date-time literals and the above usage of CDateTime. Date-time literals always use U.S. English date formats rather than settings from the locale of the particular computer on which Crystal Reports is running. Thus, the date-time literal examples above would work on all computers. On the other hand, on a French system, you could use constructions like:

```
CDateTime ("22 aout 1997") 'Same as #Aug 22, 1997#
```

Date values can be constructed with CDate and Time values with CTime:

```
CDate ("Aug 6, 1969")
CDate (1969, 8, 6) 'Specify the year, month, day
'Converts the DateTime argument to a Date
CDate (#Aug 6, 1969#)
CTime ("10:30 am")
CTime (10, 30, 0) 'Specify the hour, minute, second
CTime (#10:30 am#)
```

Range Data Types (Basic Syntax)

Ranges are designed to handle a spectrum of values. Range types are available for all the simple types except for Boolean. That is: Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range. You can generate ranges using the To, _To, To_, _To_, Is >, Is >=, Is < and Is <= keywords. In general, To is used for ranges with two endpoints, and Is is used for open ended ranges (only one endpoint). The underscores are used to indicate whether or not the endpoints are in the range.

Examples of Number Range values

The range of numbers from 2 to 5 including both 2 and 5

```
2 To 5
```

The range of numbers from 2 to 5, not including 2 but including 5

```
2 _To 5
```

All numbers less than or equal to 5

```
Is <= 5
```

All number less than 5

```
Is < 5
```

Examples of DateTime Range values

```
#Jan 5, 1999# To #Dec 12, 2000#
Is >= #Jan 1, 2000#
```

Ranges in Formulas

There are twenty-seven functions in Crystal Reports that specify date ranges. For example, the function LastFullMonth specifies a range of date values that includes all dates from the first to last day of the previous month. So if today's date is September 15, 1999 then LastFullMonth is the same as the range value CDate (#Aug 1, 1999#) To CDate (#Aug 31, 1999#).

Ranges are often used with If or Select statements. The following example computes student letter grades based on their test scores. Scores greater than or equal to 90 receive an "A", scores from 80 to 90, not including 90 receive a "B" and so on.

```
Rem Compute student letter grades
Select Case {Student.Test Scores}
   Case Is >= 90
```

```
        formula = "A"
    Case 80 To_ 90
        formula = "B"
    Case 70 To_ 80
        formula = "C"
    Case 60 To_ 70
        formula = "D"
    Case Else
        formula = "F"
End Select
```

The above example uses the Select statement which is discussed in more detail in Control Structures (Basic Syntax). You can check if a value is in a range by using the In operator. For example:

```
formula = 5 In 2 To 10 'True
formula = 5 In 2 To_ 5 'False
formula = 5 In 2 To 5 'True
```

The Maximum and Minimum functions can be used to find the endpoints of a range:

```
formula = Maximum (2 To 10) 'Returns 10
```

Array Data Types (Basic Syntax)

Arrays in Crystal Reports are ordered lists of values that are all of the same type. These values are known as the array's elements. The elements of an array can be any simple type or range type. One way to create an array is using the Array function.

Arrays are most useful when used with variables. Using variables, you can change the individual elements of an array and resize the array to accommodate more elements. This capability significantly expands the capabilities of the formula language to do complex calculations.

For example, you can accumulate database field values into a global array variable in a detail level formula, and then use a formula in a group footer to perform a calculation based on those values. This allows you to perform a wide variety of customized summary operations.

Examples

An array of three Number values. The first element is 10, the second is 5 and the third is 20.

```
Array (10, 5, 20)
```

An array of seven String values:

```
Array ("Sun", "Mon", "Tue", "Wed", "Th", "Fri", "Sat")
```

An array of two DateTime Range values:

```
Array (#Jan 1, 1998# To #Jan 31, 1998#, _
```

```
#Feb 1, 1999# To #Feb 28, 1999#)
```

You can extract individual elements out of an array using parentheses containing the index of the element you want. This is called *subscripting* the array:

```
Array (10, 5, 20) (2) 'Equal to 5
```

Note Arrays in Basic syntax are indexed from 1 (this means the first element has index 1). This is unlike in Visual Basic where arrays are indexed from 0 by default. However, in Visual Basic, arrays can be indexed from 1 by using the *Option Base* statement.

Number ranges can also be used to subscript arrays. The result is another array. For example:

```
Array (10, 5, 20) (2 To 3) 'Equal to Array (5, 20)
```

Variables (Basic Syntax)

A variable represents a specific data item, or value, and acts as a placeholder for that value. When a formula encounters a variable, the formula searches for the value of the variable and uses it in the formula. Unlike a constant value, which is fixed and unchanging, a variable can be repeatedly assigned different values. You assign a value to a variable and the variable maintains the value until you later assign a new value. Because of this flexibility, it is necessary for you to declare variables before you use them so that Crystal Reports is aware of them and understands how you intend to use them.

This section describes the key components of variables and shows you how to create variables and assign values to them.

Example

If you wanted to report on customers by area code, you could create a variable that extracts the area code from a customer fax number. The following is an example of a variable called areaCode:

```
Dim areaCode As String

areaCode = Left ({Customer.Fax}, 3)

Rem could also use: areaCode = {Customer.Fax} (1 To 3)
```

The first line of the variable example is the variable declaration; it gives the variable a name and type. The database field {Customer.Fax} is a String field and the Left function extracts the first three characters from its current value. The variable areaCode is then assigned this value.

Variable Declarations using Dim (Basic Syntax)

Before using a variable in a formula, you must declare it. A variable can hold values of a given type. The allowed types are the seven simple types (Number, Currency, String, Boolean, Date, Time and DateTime), the six range types (Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range) and variables that hold arrays of the previously mentioned types. This gives a total of 26 different types that a variable can have.

When you declare a variable, you also specify its name. A variable cannot have the same name as any function, operator or other keyword that is valid for Basic syntax. For

example, your variable cannot be named Sin, Mod or If because Sin is a built in function, Mod is a built in operator and If is a built in keyword. When typing formulas in the formula editor, the names of the built-in functions, operators, and other keywords are highlighted in a different color. This makes it easy to check if the variable name conflicts.

Once a variable is declared, it can be used in the formula. For example, you might want to assign it an initial value:

```
Dim x As Number 'Declare x to be a Number variable
x = 10 'Assign the value of 10 to x
```

You can declare more than one variable per statement by separating their declaration by commas:

```
Dim x As Number, y as String, z as DateTime Range
x = 10 : y = "hello"
z = #Jan 1, 1999# To #Jan 31, 1999#
```

Declaring variables without immediately specifying their type

In general, the type of a variable does not need to be explicitly given when declaring it. In such cases, the variable's type is determined by the first assignment that is made to it. This is similar to the special variable formula. This is different from Visual Basic in which a variable whose type is not given at declaration automatically has the Variant type. However, in practice, it means that you can write formulas in a similar style to what you would do if using a Variant in Visual Basic.

```
Dim p 'The type of p is not known yet
p = "bye" 'The type of p is now set to be String
Dim q 'The type of q is not known yet
q = Array ("hello", p) 'q is a String Array
'Error- p is a String variable and cannot hold a Number
p = 25
Dim r
'r is a Number variable, and holds the value 5
r = (10 + 5) / 3
'The types of a and c are not known yet
Dim a, b As Boolean, c
b = False
'The type of a is now set to Boolean
'and its value is False
a = b
'The type of c is now set to Number and its value is 17
c = 2 + 3 * 5
```

Examples of declaring and initializing range variables

```
Dim gradeA, quarter
'The type of gradeA is set to Number Range
gradeA = 90 To 100
'The type of quarter is set to Date Range
quarter = CDate (1999, 10, 1) To CDate (1999, 12, 31)
```

Variable Scope (Basic Syntax)

Variable scopes are used to define the degree to which variables in one formula are made available to other formulas. There are three levels of scope in Crystal Reports:

Local (Basic Syntax)

Global (Basic Syntax)

Shared (Basic Syntax)

Every variable has a scope, and this scope is specified when the variable is declared.

Local Variables (Basic Syntax)

Variables with local scope, also known as local variables, are declared using either the Dim or Local keywords.

```
Local x As Number 'equivalent to Dim x As Number
```

Local variables are restricted to a single formula and a single evaluation of that formula. This means that you cannot access the value of a local variable in one formula from a different formula.

Example

```
Rem Formula A
Local x as Number
x = 10
formula = x
Rem Formula B
EvaluateAfter ({@Formula A})
Local x as Number
formula = x + 1
```

The function call EvaluateAfter ({@Formula A}) ensures that Formula B will be evaluated after Formula A is evaluated. Formula A returns a value of 10 and Formula B returns a value of 1. Formula B does not have access to Formula A's x and thus cannot use the value of 10 and add 1; instead, it uses the default value for the uninitialized local variable x found in Formula B, which is 0, and adds 1 to it to get 1.

You can also create local variables with the same name but different types in different formulas. For example, the type declarations in formulas A and B do not conflict with:

```
Rem Formula C
Local x as String
```

```
x = "hello"

formula = x
```

Local variables are the most efficient of the three scopes. In addition, they do not interfere with one another in different formulas. For these reasons, it is best to declare variables to be local whenever possible.

Global Variables (Basic Syntax)

Global variables use the same memory block to store a value throughout the main report. This value is then available to all formulas that declare the variable, except for those in subreports. Declare a global variable as in the following example:

```
Global y As String
```

Since global variables share their values throughout the main report, you cannot declare a global variable in one formula with one type and then declare a global variable with the same name in a different formula with a different type.

When to Use Global Variables

Global variables are often used to perform complex calculations where the results of a formula depend upon the grouping and page layout of the actual printed report. This is accomplished by creating several formulas, placing them in different sections of the report, and having the different formulas interact via global variables.

Example

```
Rem Formula C

Global x as Number

x = 10

formula = x

Rem Formula D

'call the function WhileReadingRecords

WhileReadingRecords

Global x as Number

x = x + 1

formula = x
```

If Formula C is placed in the Report Header and then Formula D is placed in a detail section, Formula C will be evaluated before Formula D. Formula C will be evaluated once and then Formula D will be evaluated for each record appearing in the detail section. Formula C returns 10. For the first detail record, Formula D returns 11. This is because the value 10 of x is retained from when it was set by Formula C. Formula D then adds 1 to this value, setting x to 11 and then returns 11. For the second detail record, formula D return 12, adding 1 to the previously retained value of x which was 11. This process continues for the remaining detail records.

The call to WhileReadingRecords tells Crystal Reports to re-evaluate Formula D as it reads in each record of the report. Otherwise, since the formula does not contain any database fields, the program will evaluate it only once before reading the records from the

database. The formula will then return the value 11 instead of 11, 12, 13, ... as the successive records are processed.

If the statement x = x + 1 is replaced by x = x + {Orders Detail.Quantity}, you create the effect of a running total based on {Orders Detail.Quantity}, although it is one starting at 10 rather than 0 because of Formula C. In this case, you can omit the call to WhileReadingRecords, since it will automatically occur because the formula contains a database field.

Shared Variables (Basic Syntax)

Shared variables use the same memory block to store the value of a variable throughout the main report and all of its subreports. Thus shared variables are even more general than global variables. To use a shared variable, declare it in a formula in the main report as in the following example:

```
Shared x As Number
x = 1000
```

and declare it in a formula in the subreport as in the following example:

```
Shared x as Number
```

To use shared variables, the variable must be declared and assigned a value before it can be passed between the main report and the subreport.

Declaring Array Variables (Basic Syntax)

There are several different ways to declare array variables. The first way is to use empty parentheses and explicitly specify the type of the array:

```
'Declare x to be a Global variable
'of Number Array type
Global x () As Number
'Initialize x
x = Array (10, 20, 30)
'Declare y to be a Shared variable
'of String Range Array type
Shared y () As String Range
'Initialize y
y = Array ("A" To "C", "H" To "J")
```

The second way is to declare the variable without specifying that it is an array and without giving its type and waiting for the first assignment to the variable to completely specify its type:

```
'Declare y to be a Local variable
'but do not specify its type
Dim y
'The type of y is now set to be a String Array
```

```
y = Array ("Sun", "Mon", "Tue", "Wed", "Th", _
           "Fri", "Sat")
```

The third way is to declare that the variable is an array but not specify its type fully until the first assignment. Assuming the declaration of y above:

```
'Declare z to be a Local variable that is an Array
Local z()
'z is set to Array ("Mon", "Tue") and is a String Array
z = y(2 to 3)
```

The fourth way is to explicitly specify the size of the array during the declaration. If you use this technique, the array is automatically created and default values are used to fill the array. For example, for a Number Array, each element is initialized to 0 and for a String array each element is initialized to the empty string "". Since this type of declaration actually creates the array, you must specify its type with the As clause so that Crystal Reports knows how much storage space to reserve for the array.

```
Dim a(2) As String
'Assign a value to the first element of the array a
a(1) = "good"
a(2) = "bye"
'The & operator can be used to concatenate strings
'the formula returns the String "goodbye"
formula = a(1) & a(2)
```

Assigning Values to Elements of an Array

You can assign values to elements of an array and also use the values of the elements for other computations.

```
Global x() As String
x = Array ("hello", "bye", "again")
'Now x is Array ("hello", "once", "again")
x (2) = "once"
'The statement below would cause an error if not
'commented out since the array has size 3
'x (4) = "zap"
'The formula returns the String "HELLO"
formula = UCase (x (1))
```

The Redim and Redim Preserve keywords can be used to resize an array, which is useful if you want to add extra information to it. Redim erases the previous contents of the array first before resizing it whereas Redim Preserve preserves the previous contents.

```
Dim x () As Number
```

```
Redim x (2) 'Now x is Array (0, 0)
x (2) = 20 'Now x is Array (0, 20)
Redim x (3) 'Now x is Array (0, 0, 0)
x (3) = 30 'Now x is Array (0, 0, 30)
Redim Preserve x (4) 'Now x is Array (0, 0, 30, 0)
formula = "finished"
```

Arrays and For/Next loops

Arrays are commonly used with For/Next Loops (Basic Syntax). The following example creates and then uses the array `Array (10, 20, 30, ..., 100)` using a For/Next loop.

```
Dim b (10) As Number
Dim i
For i = 1 To 10
   b(i) = 10 * i
Next i
formula = b(2) 'The formula returns the Number 20
```

Several variables can be declared in a single statement by separating their declarations with commas.

Default Values for Simple Types (Basic Syntax)

An uninitialized variable will have the default value for its type. In general, it is not a good programming practice to rely on the default values of types. For example, initialize all local variables in your formula, initialize all global variables in a formula placed in the Report Header, and initialize all shared variables in a formula placed in the Report Header of the main report.

When an array is resized using the Redim keyword, the entries are filled with default values for the type.

Default values

Number

```
0
```

Currency

```
CCur (0)
```

String

```
"" 'The empty string
```

Date

```
CDate (0, 0, 0) 'The null Date value
```

Time

The null Time value. Value held by an uninitialized Time variable.

DateTime

The null DateTime value. Value held by an uninitialized DateTime variable.

Note It is not recommended that your formulas rely on the values of uninitialized range or array variables.

Automatic Type Conversions (Basic Syntax)

Generally in Crystal Reports, values of one type cannot be used where values of another type are expected without explicitly supplying a type conversion function. For example:

```
Dim postalCode as String
'Error- assigning a Number value to a String variable
postalCode = 10025
'OK- use the type conversion function CStr
'to create "10025"
postalCode = CStr (10025, 0)
```

However, there are a few conversions that are made automatically:

Number to Currency

Date to DateTime

Simple type to Range value of the same underlying simple type

For example, the following assignments are correct:

```
Dim cost As Currency
'Same as: cost = CCur (10)
cost = 10
Dim orderDate As DateTime
'Same as: orderDate = CDateTime (1999, 9, 23, 0, 0, 0)
orderDate = CDate (1999, 9, 23)
Dim aRange As Number Range
'Same as: aRange = 20 To 20
aRange = 20
Dim aRangeArray () As Number Range
'Same as :
'aRangeArray = Array (10 To 10, 20 To 25, 2 To 2)
aRangeArray = Array (10, 20 To 25, 2)
```

Note The opposite conversions are not allowed. For example:

```
Dim num As Number
num = 5 + CCur (10) 'Error
'OK- convert to Number type using the CDbl function
num = CDbl (5 + CCur (10))
```

5 is converted to CCur (5) and added to CCur (10) to make CCur (15). However, this Currency value cannot be automatically assigned to the Number variable num since automatic conversions from Currency to Number are not allowed. Similarly, functions accepting a Currency argument can be supplied a Number argument instead, and the Number argument will be converted to a Currency, whereas functions accepting a Number argument cannot be supplied a Currency argument without first explicitly converting the Currency to a Number using CDbl.

Functions (Basic Syntax)

Functions are built-in procedures or subroutines used to evaluate, make calculations on, or transform data. When you specify a function, the program performs the set of operations built into the function without you having to specify each operation separately.

Functions Overview (Basic Syntax)

Summary Functions (Basic and Crystal Syntax)

Date Ranges (Basic and Crystal Syntax)

Array Functions (Basic and Crystal Syntax)

Evaluation Time Functions (Basic and Crystal Syntax)

Print State Functions (Basic and Crystal Syntax)

Document Properties Functions (Basic and Crystal Syntax)

Additional Functions (Basic and Crystal Syntax)

Conditional Formatting Functions (Basic Syntax)

General Purpose Conditional Formatting Functions (Basic Syntax)

Functions Overview (Basic Syntax)

When using a function in a formula, type the name of the function and supply the arguments required. For example, the Len function requires a String argument and computes the length of the string.

```
Dim x As String

x = "hello"

formula = Len (x) 'The formula returns the Number 5
```

Supplying arguments of the incorrect type required by the function produces an error. For example, calling Len (3) would produce an error since Len does not accept a Number argument.

Functions sometimes can accept different numbers of arguments or types of arguments. For example, the CDate function could accept a single String argument to form a Date value or 3 Number values holding the year, month and day respectively and form a Date value from them.

Example with the Mid function

```
Dim x as String

x = "hello"

'Start at position 2, go to the end of the string

formula = Mid (x, 2) 'formula is now "ello"
```

```
'Start at position 2, extract 1 character

formula = Mid (x, 2, 1) 'formula is now "e"
```

The classes of functions are: Math, Summary, Financial, String, Date/Time, Date Range, Array, Type Conversion, Programming Shortcuts, Evaluation Time, Print State, Document Properties and Additional Functions. There are also some functions specific to conditional formatting formulas.

Functions Similar to Visual Basic Functions

The Math, Financial, String, Date/Time, Type Conversion and Programming Shortcuts groups consist mainly of functions that are familiar to Visual Basic users. Most of the functions are intended to work in the same way as the Visual Basic function of the same name.

Sometimes functions will have more overloads than are available in Visual Basic.

For example, the CDate function supports the Visual Basic overload of creating a Date value from a String value, such as CDate ("Sept 18, 1999") but it also supports an overload of creating a Date value by supplying the year, month and day as Number arguments e.g. CDate (1999, 9, 18). The overloads are indicated in the Functions tree.

Some functions that are supported by Basic syntax are not listed in the Basic syntax Functions tree. This is because they are equivalent to Basic syntax functions that are already listed in the tree.

For example, the Length function, which is the traditional Crystal syntax function for finding the length of a string, is not listed in the Basic Syntax functions tree because it works the same as the Len function.

Summary Functions (Basic and Crystal Syntax)

The Summary function group provides functions for creating summary fields such as:

```
Sum({Orders.Order Amount}, {Orders.Ship Via})
```

Summary fields are normally created using the Insert Summary or Insert Grand Total dialogs. Alternatively, you can create a summary field exclusively for use by your formula by filling in the arguments to one of the functions in the Summary functions section. However, any groups that refer to summary fields must already exist in the report.

Date Ranges (Basic and Crystal Syntax)

Date ranges produced by these functions depend on the current date. For example, if today's date is September 18, 2000, then LastFullMonth is the Date Range value:

```
CDate(#Aug 1, 2000#) To CDate(#Aug 31, 2000#)
```

This functionality is often useful, but if you want to determine a date range based on a database field such as {Orders.Order Date}? The Date/Time functions can be used instead.

Basic Syntax Example

```
Dim d As Date

d = CDate ({Orders.Order Date})

Dim dr As Date Range
```

```
dr = DateSerial (Year(d), Month(d) - 1, 1) To _
     DateSerial (Year(d), Month(d), 1 - 1)
'At this point dr is the Date Range value holding
'the last full month before {Orders.Order Date}
```

Crystal Syntax Example

```
Local DateVar d := CDate ({Orders.Order Date});
Local DateVar Range dr;
dr := DateSerial (Year(d), Month(d) - 1, 1) To
      DateSerial (Year(d), Month(d), 1 - 1);
//At this point dr is the Date Range value holding
//the last full month before {Orders.Order Date}
```

The DateSerial function makes this easy because you don't have to worry about special cases. It never lets you create an invalid date. For example, DateSerial (1999, 1 - 1, 1) is December 1, 1998. Note that in the above example, {Orders.Order Date} is actually a DateTime field and so the CDate function is used to convert it to a date by truncating the time part.

Array Functions (Basic and Crystal Syntax)

The array functions compute summaries of an array's elements. For example, the Sum function when applied to an array returns the sum of the elements of the array.

Basic Syntax Example

The following formula returns 100:

```
formula = Sum (Array (10, 20, 30, 40))
```

Crystal Syntax Example

The following formula returns 100:

```
Sum ([10, 20, 30, 40])
```

Evaluation Time Functions (Basic and Crystal Syntax)

These are the report specific functions: **BeforeReadingRecords**, **WhileReadingRecords**, **WhilePrintingRecords** and **EvaluateAfter**. You can use these functions to guide Crystal Reports as to when your formula should be evaluated.

Should the formula be evaluated before retrieving the records from the database, while reading the records from the database but before the records have been grouped, sorted and summarized, or while printing the report, when the records are grouped, sorted and summarized? In general, Crystal Reports sets an appropriate evaluation time for your formula, based on how much information the formula needs. For example, if a formula uses a database field, then it cannot be evaluated before the records are read from the database. However, you sometimes need to force a later evaluation time than normal to get the desired effect..

Normally, the returned value of a function is used further in a formula. However, evaluation time functions are called to change the internal behavior of Crystal Reports and their return value is not used. They can be called by just placing their name in a separate statement, optionally preceded by the keyword Call.

```
WhilePrintingRecords
```

```
Call WhilePrintingRecords
```

Print State Functions (Basic and Crystal Syntax)

These functions are reporting-specific functions that deal with the state of a report being previewed.

For example, the notation `{Orders.Order Date}` refers to the value of the field in the current record where `PreviousValue ({Orders.Order Date})` refers to the value in the immediately preceding record. `NextValue ({Orders.Order Date})` refers to the value in the next record. `IsNull ({Orders.Order Date})` checks if the field's value is null.

Other examples are **PageNumber** and **TotalPageCount**. These can be used to access pagination information about your report.

Document Properties Functions (Basic and Crystal Syntax)

These functions return values of attributes pertaining to a document. For example, **PrintDate** and **ReportTitle**.

Additional Functions (Basic and Crystal Syntax)

These are functions that are in User Function Libraries (UFLs). A UFL is a separate dynamic link library or Automation server that you create and Crystal Reports uses to add your own customized functions to the formula language. Writing a UFL is more involved than writing a formula using Basic or Crystal syntax.

> **Note** Using UFLs makes your reports less portable because you must distribute your UFL along with the report.

Conditional Formatting Functions (Basic Syntax)

When writing a conditional formatting formula, you may want to use the additional functions that appear at the top of the Functions tree.

Example

If you wanted to format the {Customer.Last Year's Sales} field so that sales of more than $100,000 are printed in green and sales of less than $15,000 are printed in red and all else are printed in black.

```
Rem Conditional formatting example 1
If {Customer.Last Year's Sales} > 100000 Then
   formula = crGreen
ElseIf {Customer.Last Year's Sales} < 15000 Then
   formula = crRed
```

```
Else
   formula = crBlack
End If
```

Since this is a font color formatting function, the list of Color Constants appears in the Functions tree. This example uses three: crGreen, crRed and crBlack. You could have used the actual numeric values of the color constants instead. For example, crRed is 255 and crGreen is 32768. However, the formula is easier to understand using the color constants. All constant functions in Basic syntax have the "cr" prefix.

Note Some formatting attributes do not use constant functions. For example, if you wanted to not print {Customer.Last Year's Sales} values if the sales were less than $50,000, you could write the following conditional formatting formula for the suppress attribute:

```
Rem Conditional formatting example 2
If {Customer.Last Year's Sales} < 50000 Then
   formula = True 'suppress the value
Else
   formula = False 'do not suppress the value
End If
```

Or more simply:

```
Rem Conditional formatting example 3 -
Rem equivalent to example 2
formula = {Customer.Last Year's Sales} < 50000
```

If the last year's sales are less than $50,000, then the expression

```
{Customer.Last Year's Sales} < 50000
```

is True, and so the formula returns True. On the other hand, if the last year's sales are greater than or equal to $50,000, then

```
{Customer.Last Year's Sales} < 50000
```

is False and so the formula returns False.

General Purpose Conditional Formatting Functions (Basic Syntax)

There are three general purpose conditional formatting functions:

CurrentFieldValue

DefaultAttribute

GridRowColumnValue

These functions are displayed at the top of the Functions tree whenever appropriate.

DefaultAttribute can be used for any formatting formula, **CurrentFieldValue** for any formatting formula where you are formatting a field value, and **GridRowColumnValue** for any formatting formula where you are formatting a field value in a Cross-Tab or OLAP grid.

CurrentFieldValue enables you to conditionally format Cross-Tab or OLAP grid cells based on their value. **GridRowColumnValue** enables you to conditionally format the cells of a Cross-Tab or OLAP grid based on row or column headings values. These two functions are essential in some situations as there is no other way in the formula language to refer to these fields.

Example

If you wanted Cross-Tab cells to be suppressed if the values are less than 50,000:

```
Rem Conditional formatting example 4
formula = CurrentFieldValue < 50000
```

Operators (Basic Syntax)

Operators are special symbols or words that describe an operation or an action to take place between two or more values. The program reads the operators in a formula and performs the actions specified.

Arithmetic Operators (Basic Syntax)

Comparison Operators (Basic Syntax)

Boolean Operators (Basic Syntax)

Null Fields and Null Values (Basic Syntax)

Arithmetic Operators (Basic Syntax)

Arithmetic operators are used to combine numbers, numeric variables, numeric fields and numeric functions to get another number.

The arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), integer division (\), modulus (Mod), negation (-) and exponentiation (^).

Examples

```
'Outstanding preferred stock as a percent of
'common stock
formula = ({Financials.Preferred Stock} / _
           {Financials.Common Stock}) * 100
'The square root of 9, Sqr(9), is 3.
'The formula returns 17.
formula =  7 + 2 * 3 - 2 + Sqr(6 + 3) * Len("up")
```

Order of Precedence

In general, the program evaluates expressions in the following order:

- from left to right
- follows the rules of precedence from basic math

The arithmetic operators in Crystal Reports have the same order of precedence as in Visual Basic. Here is the list, from highest precedence to lowest:

- Exponentiation (^)
- Negation (-)

Multiplication and division (*, /)

Integer Division (\)

Modulus (Mod)

Addition and subtraction (+, -)

Example

Multiplication and division are performed first from left to right. Then addition and subtraction are performed. For example, 5 + 10 * 3 = 5 + 30 = 35.

You can change this order of precedence by using parentheses. For example, (5 + 10) * 3 = 15 * 3 = 45. If you are unsure of the order of precedence, it is a good idea to clarify your intentions with parentheses.

Comparison Operators (Basic Syntax)

Comparison operators are usually used to compare operands for a condition in a control structure such as an If statement.

The comparison operators are equal (=), not equal (<>), less than (<), less than or equal to (<=), greater than (>) and greater than or equal to (>=).

Comparison operators as a group all have lower precedence than the arithmetic operators. For example, expressions like 2 + 3 < 2 * 9 are the same as (2 + 3) < (2*9).

Boolean Operators (Basic Syntax)

Boolean operators are typically used with comparison operators to generate conditions for control structures.

The Boolean operators are, in order of precedence from greatest to lowest: Not, And, Or, Xor, Eqv and Imp.

Boolean operators as a group have lower precedence than the comparison operators. Thus for example, the expression 2 < 3 And 4 >= -1 is the same as (2 < 3) And (4 >= -1).

Null Fields and Null Values (Basic Syntax)

In general, when Crystal Reports encounters a null valued field in a formula, it immediately stops evaluating the formula and produces no value. If you want to handle null field values in your formula, you must explicitly do so using one of the special functions designed for handling them: IsNull, PreviousIsNull or NextIsNull.

Relating to operators, when Crystal Reports evaluates the condition:

```
IsNull({Product.Color}) Or _
InStr({Product.Color}, " ") = 0
```

It first evaluates IsNull ({Product.Color}), and when it determines that this is True, it knows that the whole condition is True, and does not need to check whether

```
InStr({Product.Color}, " ") = 0
```

In other words, Crystal Reports will stop evaluating a Boolean expression when it can predict the results of the whole expression. In the following example, the formula guards against attempting to divide by zero in the case that denom is 0:

```
Dim num As Number, denom As Number
```

```
...
If denom <> 0 And num / denom > 5 Then
...
```

Note Visual Basic does not support this technique, since all parts of a Boolean expression in Visual Basic are evaluated, even if not necessary.

Example

The {Product.Color} field contains both basic colors such as "red" and "black" and more descriptive two word colors such as "steel satin" and "jewel green". Here's an example of a formula that writes out "basic" for the basic colors and "fancy" for the others.

```
If InStr({Product.Color}, " ") = 0 Then
    formula = "basic"
Else
    formula = "fancy"
End If
```

The function call to InStr searches the {Product.Color} string for a space. If it finds a space, it returns the position of the space, otherwise it returns 0. Since basic colors are only one word with no spaces, InStr will return 0 for them.

For some products, such as the Guardian Chain Lock, a color value was not recorded and so the {Product.Color} field has a null value in the database for that record. Thus, the Guardian Chain Lock record does not have any word printed beside it.

Here is an example of how to fix the above example using IsNull:

```
If IsNull({Product.Color}) Or _
   InStr({Product.Color}, " ") = 0 Then
   formula = "basic"
Else
   formula = "fancy"
End If
```

Control Structures (Basic Syntax)

Formulas without control structures execute each statement in the formula only once. When this happens the formula is evaluated. The statements are executed in a sequential fashion, from the first statement in the formula to the last. Control structures enable you to vary this rigid sequence. Depending upon which control structure you choose, you can skip over some of the statements or repeatedly evaluate some statements depending on certain conditions. Control structures are the primary means of expressing business logic and typical report formulas make extensive use of them.

Basic syntax supports many of the main control structures from Visual Basic with the same syntax. One of the advantages of the Basic language is it is easy to read block notation for control structures. This simplifies the writing and debugging of complex formulas.

If Statements (Basic Syntax)

The If statement is one of the most useful control structures. It enables you to evaluate a sequence of statements if a condition is true and evaluate a different sequence of statements if it is not true.

> **Note** When formatting with conditional formulas, always include the *Else* keyword; otherwise, values that don't meet the If condition may not retain their original format. To prevent this, use the DefaultAttribute function (If...Else formula = DefaultAttribute).

Example

A company plans to pay a bonus of 4 percent to its employees except for those who work in Sales who will receive 6 percent. The following formula using an If statement would accomplish this:

```
Rem Multi-line If example 1
If {Employee.Dept} = "Sales" Then
   formula = {Employee.Salary} * 0.06
Else
   formula = {Employee.Salary} * 0.04
End If
```

In this example, if the condition {Employee.Dept} = "Sales" evaluates as true, then the

```
formula = {Employee.Salary} * 0.06
```

statement is processed. Otherwise the statement following the Else, namely the

```
formula = {Employee.Salary} * 0.04
```

is processed.

Suppose another company wants to give employees a 4% bonus, but with a minimum bonus of $1,000. Notice that the Else clause is not included; it is optional, and not needed in this case.

```
Rem Multi-line If example 2
formula = {Employee.Salary} * 0.04
If formula < 1000 Then
   formula = 1000
End If
```

Now suppose that the previous company also wants a maximum bonus of $5,000. You now need to use an ElseIf clause. Notice that ElseIf is all one word. The following example has only one ElseIf clause, but you can add as many as you need.

> **Note** There is a maximum of one Else clause per If statement.

The Else clause is executed if none of the If or ElseIf conditions are true.

```
Rem Multi-line If example 3
formula = {Employee.Salary} * 0.04
If formula < 1000 Then
```

```
   formula = 1000
ElseIf formula > 5000 Then
   formula = 5000
End If
```

Example

Suppose that a company wants to compute an estimate of the amount of tax an employee needs to pay and write a suitable message. Income below $8,000 is not taxed, income between $8,000 and $20,000 is taxed at 20%, income between $20,000 and $35,000 is taxed at 29%, and income above $35,000 is taxed at 40%.

```
Rem Multi-line If example 4
Dim tax As Currency, income As Currency
income = {Employee.Salary}
Dim message As String
If income < 8000 Then
   tax = 0
   message = "no"
ElseIf income >= 8000 And income < 20000 Then
   message = "lowest"
   tax = (income - 8000)*0.20
ElseIf income >= 20000 And income < 35000 Then
   message = "middle"
   tax = (20000 - 8000)*0.20 + (income - 20000)*0.29
Else
   message = "highest"
   tax = (20000 - 8000)*0.20 + (35000 - 20000)*0.29 + _
         (income - 35000)*0.40
End If
Dim taxStr As String
Rem use 2 decimal places
Rem and use the comma as a thousands separator
taxStr = CStr (tax, 2, ",")
formula = "You are in the " & message & _
          " tax bracket. " & _
          "Your estimated tax is " & taxStr & "."
```

Notice, the use of variables to simplify the logic of the computation. Also, notice that there are two statements that are executed when one of the conditions are met; one assigns the

tax variable, and the other assigns the message variable. It is often useful to have multiple statements executed as a result of a condition.

Single-Line and Multi-Line If Statements (Basic Syntax)

There are two kinds of If statement, the single-line if statement and the multi-line if statement. Starting on a new line after the first *Then* turns your If statement into a multi-line If statement. Otherwise it is a single-line If statement. The multi-line If statement always includes an End If whereas the single line If statement does not.

> **Note** Because of the use of line-continuation characters, single-line If statements do not need to be on a single line. In general, it is preferable to use multi-line If statements since they have a clearer layout. However, for simple situations, the single-line If statement is sometimes used.

```
Rem Single-line If example 1
Rem Same result as multi-line If example 1
If {Employee.Dept} = "Sales" Then _
   formula = {Employee.Salary} * 0.06 _
Else _
   formula = {Employee.Salary} * 0.04
```

Here is an example showing various forms of single-line If statements:

```
Rem Single-line If example 2
Dim per As Number, extra As Boolean
per = 2 : extra = False
'An example with no Else clause
If {Employee.Dept} = "Sales" Then per = 10
'More than 1 statement in the Then or Else part can
'be included by separating them with colons
If {Employee.Dept} = "R&D" Then _
   per = 5 : extra = True _
Else _
   per = 3
```

Select Statements (Basic Syntax)

The Select statement is similar to an If statement. Sometimes however, you can write formulas that are clear and less repetitive using the Select statement. This example evaluates the {Customer.Fax} field to determine if the area code is for Washington state (206, 360, 509) or British Columbia, Canada (604, 250):

```
Rem Select example 1
Select Case Left ({Customer.Fax}, 3)
```

```
   Case "604", "250"
      formula = "BC"
   Case "206", "509", "360"
      formula = "WA"
End Select
```

The expression right after the Select Case keywords is called the Select condition. In the above example it is Left ({Customer.Fax}[1 To 3]). The Select statement tries to find the first Case that matches the Select condition, and then executes the statements following it, up until the next Case.

```
Rem Same effect as Select example 1
Dim areaCode As String
areaCode = Left ({Customer.Fax}, 3)
If areaCode In Array ("604", "250") Then
   formula = "BC"
ElseIf areaCode In Array ("206", "509", "360") Then
   formula = "WA"
End If
```

Example

This formula groups the number of Oscar nominations a movie received into low, medium, high or extreme categories and in the process, shows some of the possibilities for the expression lists following the Case labels. Notice the optional Case Else clause. If none of the Case expression lists are matched by the preceding Case clauses, then the Case Else clause is matched. For example, in the following example, if {movie.NOM} is 11, then the formula returns "extreme".

```
Rem Select example 2
Select Case {movie.NOM}
   Case 1,2,3, Is < 1
      Rem Can have multiple statements in the
      Rem statement blocks
      formula = "low"
   Case 4 To 6, 7, 8, 9
      formula = "medium"
   Case 10
      formula = "high"
   Case Else
      formula = "extreme"
End Select
```

For/Next Loops (Basic Syntax)

For/Next loops enable you to evaluate a sequence of statements multiple times. This is unlike the If and Select statements where the program passes through each statement at most once during the formula's evaluation.

For/Next loops are best when you know the number of times that the statements needs to be evaluated in advance.

For Loop Syntax

Example 1

Suppose you want to reverse the {Customer.Customer Name} string. For example, "City Cyclists" becomes "stsilcyC ytiC".

```
Rem Reverse a string version 1
formula = ""
Dim strLen
strLen = Len ({Customer.Customer Name})
Dim i
For i = 1 To strLen
   Dim charPos
   charPos = strLen - i + 1
   formula = formula & _
             Mid({Customer.Customer Name}, charPos, 1)
Next i
```

Examine how this formula works assuming that the current value of the field {Customer.Customer Name} is "Clean Air". The variable strLen is assigned to be the length of "Clean Air", namely 9. At this time it is also typed to be a Number variable. The variable i is known as a For counter variable since its value changes with each iteration of the For loop. In other words, it is used to count the iterations of the loop. The For loop will iterate 9 times, during the first time, i is 1, then i is 2, then i is 3 and so on until finally i = 9. During the first iteration, the ninth character of {Customer.Customer Name} is appended to the empty special variable formula. As a result formula equals "r" after the first iteration. During the second iteration, the eighth character of {Customer.Customer Name} is appended to formula and so formula equals "ri". This continues until after the ninth iteration, formula equals, "riA naelC" which is the reversed string.

Example 2

Here is a simpler version of the above formula that uses a Step clause with a negative Step value of -1. For the "Clean Air" example, i is 9 for the first iteration, 8 for the second, 7 for the third and so on until it is 1 in the final iteration.

```
Rem Reverse a string version 2
formula = ""
Dim i
```

```
For i = Len ({Customer.Customer Name}) To 1 Step -1
   formula = formula + _
             Mid({Customer.Customer Name}, i, 1)
Next i
```

Example 3

The simplest version is to use the built in function StrReverse:

```
Rem Reverse a string version 3
formula = StrReverse ({Customer.Customer Name})
```

The built in String functions in Crystal Reports can handle many of the string processing applications which would traditionally be handled using a For/Next loop or some other kind of loop. However, For/Next loops provide the most flexibility and power in processing strings and arrays. This can be essential if the built-in functions do not cover your intended application.

For/Next Loop Example (Basic Syntax)

Here is a more detailed example of Crystal Reports' string processing capabilities. The Caesar cipher is a simple code that is traditionally credited to Julius Caesar. In this code, each letter of a word is replaced by a letter five characters further in the alphabet. For example, "Jaws" becomes "Ofbx". Notice that "w" is replaced by "b". Since there are not 5 characters after "w" in the alphabet, it starts again from the beginning.

Here is a formula that implements applying the Caesar cipher to the field {Customer.Customer Name} in the Xtreme database:

```
Rem The Caesar cipher
Dim inString 'The input string to encrypt
inString = {Customer.Customer Name}
Dim shift
shift = 5
formula = ""
Dim i
For i = 1 To Len(inString)
   Dim inC, outC
   inC = Mid(inString, i, 1)
   Dim isChar, isUCaseChar
   isChar = LCase(inC) In "a" To "z"
   isUCaseChar = isChar And (UCase (inC) = inC)
   inC = LCase(inC)
   If isChar Then
      Dim offset
```

```
      offset = (Asc(inC) + shift - Asc("a")) Mod _
               (Asc("z") - Asc("a") + 1)
      outC = Chr(offset + Asc("a"))
      If isUCaseChar Then outC = UCase(outC)
   Else
      outC = inC
   End If
   formula = formula & outC
Next i
```

In the above example, there is a multi-line If statement nested within the statements block of the For/Next loop. This If statement is responsible for the precise details of shifting a single character. For example, letters are treated differently from punctuation and spaces. In particular, punctuation and spaces are not encoded. Control structures can be nested within other control structures and multiple statements can be included in the statement block of a control structure.

Exiting from For/Next Loops (Basic Syntax)

You can exit from a For/Next loop by using the Exit For statement. The following example searches the Global array names for the name "Fred". If it finds the name, it returns the index of the name in the array. Otherwise it returns -1. For example, if the names array is:

```
Array ("Frank", "Helen", "Fred", "Linda")
```

Then the formula returns 3.

```
Global names () As String
'The names array has been initialized and filled
'in other formulas
Dim i
formula = -1
'The UBound function returns the size of its array
'argument
For i = 1 to UBound (names)
   If names (i) = "Fred" Then
      formula = i
      Exit For
   End If
Next I
```

Do Loops (Basic Syntax)

A Do loop can be used to execute a fixed block of statement an indefinite number of times.

The 4 different types of Do loops

Type of Do Loop	Explanation	Example

Do While … Loop	The Do While ... Loop evaluates the condition, and if the condition is true, then it evaluates the statements following the condition.
	When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the statements again.
	It continues repeating this process until the condition is false.
		`Do While condition`
		`  statements`
		`Loop`

Do Until ... Loop	The Do Until ... Loop is similar to the Do While ... Loop except it keeps evaluating the statements until the condition is true rather than while it is true.	`Do Until condition`
		`  statements`
		`Loop`

Do ... Loop While	The Do ... Loop While evaluates the statements only once.
	It then evaluates the condition, and if the condition is true, evaluates the statements again. This process continues until the condition is false.
		`Do`
		`  statements`
		`Loop While condition`

Do ... Loop Until	Similar to Do ... Loop While except that it evaluates the statements until the condition is true.	`Do`
		`  statements`
		`Loop Until condition`

Note The Do loops support an Exit Do statement to immediately jump out of the loop. The Exit Do statement is similar to the Exit For in For/Next loops.

Do While … Loop Formula Example

The following example searches for the first occurrence of a digit in an input string. If a digit is found, it returns its position, otherwise it returns -1. In this case, the input string

is set explicitly to a string constant, but it could be set equal to a String type database field instead.

For example, for the input String, "The 7 Dwarves", the formula returns 5, which is the position of the digit 7.

```
Dim inString

inString = "The 7 Dwarves"

Dim i, strLen

i = 1

strLen = Len (inString)

formula = -1

Do While i <= strLen And formula = -1

   Dim c As String

   c = Mid (inString, i, 1)

   If IsNumeric (c) Then formula = i

   i = i + 1

Loop
```

While Loops (Basic Syntax)

The While loop is similar to the Do While ... Loop except that it does not support an Exit statement. It uses While ... Wend instead of Do While ... Loop as its syntax.

```
While condition

   statements

Wend
```

Preventing Infinite Loops (Basic Syntax)

There is a safety mechanism to prevent report processing from hanging due to an infinite loop. Any one evaluation of a formula can have at most 100,000 loop condition evaluations per formula evaluation. For example:

```
Dim i

i = 1

Do While i <= 200000

   If i > {movie.STARS} Then Exit Do

   i = i + 1

Loop

formula = 20
```

If {movie.STARS} is greater than 100,000 then the loop condition (i <= 200000) will be evaluated more than the maximum number of times and an error message is displayed. Otherwise the loop is OK.

Note The safety mechanism applies on a per formula base, not for each individual loop. For example:

```
Dim i
i = 1
For i = 1 To 40000
   formula = Sin (i)
Next i
Do While i <= 70000
   i = i + 1
Loop
```

The above formula also triggers the safety mechanism since the 100,000 refers to the total number of loop condition evaluations in the formula and this formula will have 40001 + 70001 such evaluations.

Crystal Syntax

When creating formulas, you have the option of using either Crystal or Basic syntax. Almost any formula written with one syntax can be written with the other. Reports can contain formulas that use Basic syntax as well as formulas that use Crystal syntax, but a single formula can use only one syntax.

For descriptions and examples of individual functions and operators, go to http://support.businessobjects.com/search/ and search for *cr8_formularef.zip*.

Crystal Syntax Fundamentals

The Result of a Formula

The result of a formula, or the value that is printed when the formula is placed in a report, is called the *value returned by the formula*. Every formula in Crystal Reports must return a value. For example, here is a simple Crystal syntax formula that returns a value of 10:

```
10
```

The value returned by a formula can be one of the seven simple data types supported. These are Number, Currency, String, Boolean, Date, Time, and DateTime.

Note Crystal Reports also supports range types and array types, but these cannot be returned by a formula.

For example, suppose a company has a shipping policy in which orders over $1,000 are insured, but orders below that amount are not insured:

```
//A formula that returns a String value
If {Orders.Order Amount} >= 1000 Then
   "Insured shipping"
Else
   "Regular shipping"
```

The formula returns the text string value "Insured shipping" if the value of the database field {Orders.Order Amount} is greater than or equal to 1000; otherwise, it returns the text string value "Regular Shipping" otherwise.

Expression-Based Syntax

A Crystal syntax formula consists of a sequence of expressions. An expression is any combination of keywords, operators, functions, and constant values that result in a value of a given type. The value of the final expression is the value returned by the formula and what gets printed. Each expression must be separated from the previous expression by a semicolon (;).

The fact that a Crystal syntax formula is a sequence of expressions whose result is the value of the final expression is the most important concept in understanding Crystal syntax. This expression-based syntax allows you to write very short formulas with a lot of functionality.

Case-Sensitivity

All variable names, functions, and keywords used in a Crystal syntax formula are not case-sensitive. For example, the keyword *Then* could equivalently be typed in as *then* or *THEN*.

The only exception to this rule is for strings. The string "Hello" is not the same as the string "hello".

Practice using the Xtreme Sample Database

Many of the examples in this section refer to the Xtreme sample database.

Note For information on configuring this database and its ODBC entry, see Appendix: Location of Xtreme Sample Database and ODBC DSN Entry for Xtreme Sample Database.

Comments (Crystal Syntax)

Formula comments are notes included with a formula to explain its design and operation. Comments do not print and they do not affect the formula; they appear only in the Formula Editor. Use comments to explain the purpose of a formula or explain the steps involved in writing it.

Comments begin with two forward slashes (//) and are followed by the text of the comment. Everything that follows the slashes on the same line is treated as being part of the comment:

```
//This formula returns the string "Hello"

//This is another comment

"Hello" //Comments can be added at the end of a line

//Comments can occur after the formula text
```

Fields (Crystal Syntax)

Many of the fields you use when creating your report can also be referred to in your formulas. For example, database, parameter, running total, SQL expression, summary,

and group name fields can all be used in a formula. You can also refer to other formula fields in your formula.

The easiest way to insert a field into your report is to double-click a field's name in the Report Fields tree. This ensures that the correct syntax for the field is used.

How fields appear in formulas

Database, parameter, formula, running total and SQL expression fields have their names surrounded by braces.

Database field names are taken from the database: `{Employee.Last Name}`

Parameter, formula, running total, and SQL expression field names are specified when the fields are created.

- parameter fields also includes a question mark: `{?my parameter field}`
- formula fields include an at sign: `{@another formula}`
- running totals fields include a pound sign: `{#my running total}`
- SQL expression fields include a percent sign: `{%my SQL expression}`

Summary and group name fields look like function calls. However, they are really shorthand notation for a report field.

- sum summary field: `Sum({Orders.Order Amount}, {Orders.Ship Via})`
- group name field: `GroupName({Orders.Ship Via})`

Example

The formula in this example uses the Xtreme database. To find out how many days it takes to ship the product from the date when the order was placed, you can just subtract the ship date database field from the order date database field:

```
//A formula that uses database fields
{Orders.Ship Date} - {Orders.Order Date}
```

To find the total dollar amount of a given product that was ordered, multiply its unit price by the quantity ordered:

```
{Orders Detail.Unit Price} * {Orders Detail.Quantity}
```

To calculate a sale price of 80 percent of the original unit price:

```
{Orders Detail.Unit Price} * 0.80
```

Expressions (Crystal Syntax)

An expression is any combination of keywords, operators, functions, and constant values that result in a value of a given type. For example:

```
//An expression that evaluates to the Number value 25
10 + 20 - 5
//An expression that evaluates to the String value
//"This is a string."
"This is a string."
```

A Crystal syntax formula consists of a sequence of expressions. The value of the final expression is the value returned by the formula and what gets printed. Each expression must be separated from the previous expression by a semicolon (;).

Multiple Expressions (Crystal Syntax)

Typically, each expression takes one line, but you can continue an expression onto the next line if you need more space.

The formula below consists of five expressions. It returns the Number value 25 since that is the value of the last expression in the formula.

Example

```
//Expressions example
//The first expression. Its value is the Number
//value 30
10 + 20;
//The second expression. Its value is the String
//"Hello World". It takes up two lines.
"Hello " +
"World";
//The third expression. Its value is of Number type
{Orders Detail.Quantity} * 2 - 5;
//The fourth expression. Its value is of String type
If {Orders Detail.Quantity} > 1 Then
   "multiple units"
Else
   "one unit";
//The fifth and final expression. Its value is the
//Number value 25
20 + 5
```

Placing a semicolon after the last expression in the formula is also allowed, but is optional. For example, the above formula could have ended with:

```
20 + 5;
```

Some of the sample formulas in the Expressions (Crystal Syntax) section do not have semicolons. This is because they consist of a single expression, and a semicolon is optional after the last expression. Many formulas in Crystal syntax can be written as a single expression.

Notice that there is no semicolon after the "multiple units" string. In fact, if you put a semicolon there, the program will report an error. This is because a semicolon separates expressions, and the

```
Else
```

```
"one unit";
```

is not a separate expression. It does not stand alone apart from the If. In fact, it is an integral part of the If expression since it describes the value that the If expression will return under certain circumstances.

Note The example is not a practical example because the first four expressions in the formula did not have any effect on the last expression.

How Earlier Expressions Affect Later Expressions

The fact that a Crystal syntax formula is a sequence of expressions whose result is the value of the final expression is the most important concept in understanding Crystal syntax. This expression-based syntax allows you to write very short formulas with a lot of functionality.

Example

```
//First expression. It declares the Number variable x
//and then returns the value of an uninitialized
//Number variable, which is 0.
NumberVar x;
//Second expression. It assigns the value of 30 to x,
//and returns 30.
x := 30
```

The above formula would give an error if the first expression were omitted. This is because the second expression refers to the Number variable x, and the program needs to have x declared before it understands expressions involving it.

In general, you use variables to get the earlier expressions in a formula to affect the final expression.

Assignment (Crystal Syntax)

The assignment operator is a colon followed by an equal sign (:=).

Example

```
//Assign the Number value of 10 to the variable x
x := 10;
//Assign the String value of "hello" to the
//variable named greeting
greeting := "hello";
```

The equality operator (=) is used to check when two values are equal. A common error is to use the equality operator when the assignment operator is actually intended. This can generate a mysterious error message or no error message at all since it is often syntactically correct to use the equality operator. For example:

```
greeting = "hello";
```

The above formula checks if the value held by the variable greeting is equal to the value "hello". If it is, then the expression's value is True, and if is not then the expression's value

is False. In any case, it is a perfectly correct Crystal syntax expression (assuming that the greeting is a String variable).

Simple Data Types (Crystal Syntax)

The simple data types in Crystal Reports are

Number (Crystal Syntax)

Currency (Crystal Syntax)

String (Crystal Syntax)

Boolean (Crystal Syntax)

Date, Time, and DateTime (Crystal Syntax)

Number (Crystal Syntax)

Enter numbers without any comma separators or currency symbols. (Generally, you would want to have formatted numbers appearing as the result of a formula and not in the formula itself.)

Example

```
10000
-20
1.23
```

Currency (Crystal Syntax)

Use the dollar sign ($) to create a Currency value.

Example

```
$10000
-$20
$1.23
```

You can also use the CCur function. The initial C in CCur stands for convert and it can be used to convert Number values to Currency values:

```
CCur (10000)
CCur (-20)
CCur (1.23)
```

String (Crystal Syntax)

Strings are used to hold text. The text must be placed between double quotation marks (") or apostrophes (') and cannot be split between lines. If you want to include double quotation marks in a string delimited by double quotation marks, use two consecutive double quotation marks. Similarly, if you want to include an apostrophe in a string delimited by apostrophes, use two consecutive apostrophes.

Example

```
"This is a string."
```

```
"123"
"The word ""hello"" is quoted."
'This is also a string.'
'123'
'Last Year''s Sales'
```

If you use double quotes for the left side of the string, you must use double quotes on the right side. Similarly for apostrophes. The following example is incorrect:

```
'Not a valid string."
```

You can extract individual elements or substrings from a string by specifying the character position or a range of character positions. Negative values are allowed; they specify the position starting from the end of the string.

```
"hello" [2] //Equal to "e"
"hello" [-5] //Equal to "h"
"604-555-1234" [1 to 3] //Equal to "604"
"abcdef" [-3 to -1] //Equal to "def"
```

You can also extract substrings from a string using the Left, Right and Mid functions.

Boolean (Crystal Syntax)

The valid Boolean values are:

```
True
False
```

Note *Yes* can be used instead of True and *No* instead of False.

Date, Time, and DateTime (Crystal Syntax)

The DateTime type can hold date-times, dates only, or times only. The Date type holds dates only and the Time type holds times only. The Date and Time types are more efficient than the DateTime type, and so can be used in situations where the added functionality and flexibility of the DateTime type is not needed.

You can create DateTime values directly using the date-time literal construction. It is formed by typing in the date-time between two pound (#) signs. Many different formats are supported.

Note Date-time literals cannot be split between lines.

Examples

```
#8/6/1976 1:20 am#
#August 6, 1976#
#6 Aug 1976 13:20:19#
#6 Aug 1976 1:30:15 pm#
#8/6/1976#
#10:20 am#
```

Even though #10:20 am# looks like it could have the Time type and #8/6/1976# looks like it could have the Date type, they do not. They both have the DateTime type, as do all date-time literals. For example, you can think of #10:20 am# as a DateTime value with a null date part. To convert it to the Time type use CTime (#10:20 am#).

Instead of using date-time literals, you can use CDateTime to convert a String to a DateTime. For example,

```
CDateTime ("8/6/1976 1:20 am")
```

```
CDateTime ("10:20 am")
```

However, there is one key difference between using date-time literals and the above usage of CDateTime. Date-time literals always use U.S. English date formats rather than settings from the locale of the particular computer on which Crystal Reports is running. Thus, the date-time literal examples above would work on all computers. On the other hand, on a French system, you could use constructions like:

```
CDateTime ("22 aout 1997") //Same as #Aug 22, 1997#
```

Date values can be constructed with CDate and Time values with CTime:

```
CDate ("Aug 6, 1969")
```

```
CDate (1969, 8, 6) //Specify the year, month, day
```

```
//Converts the DateTime argument to a Date
```

```
CDate (#Aug 6, 1969#)
```

```
CTime ("10:30 am")
```

```
CTime (10, 30, 0) //Specify the hour, minute, second
```

```
CTime (#10:30 am#)
```

Range Data Types (Crystal Syntax)

Ranges are designed to handle a spectrum of values. Range types are available for all the simple types except for Boolean. That is: Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range. You can generate ranges using the To, _To, To_, _To_, UpTo, UpTo_, UpFrom, and UpFrom_ keywords. In general, To is used for ranges with two endpoints, and UpTo and UpFrom are used for open ended ranges (only one endpoint). The underscores are used to indicate whether or not the endpoints are in the range.

Examples of Number Range values

The range of numbers from 2 to 5 including both 2 and 5

```
2 To 5
```

The range of numbers from 2 to 5, not including 2 but including 5

```
2 _To 5
```

All numbers less than or equal to 5

```
UpTo 5
```

All number less than 5

```
UpTo_ 5
```

Examples of DateTime Range values:

```
#Jan 5, 1999# To #Dec 12, 2000#
```

```
UpFrom #Jan 1, 2000#
```

Using Ranges in Formulas

There are twenty-seven functions in Crystal Reports that specify date ranges. For example, the function LastFullMonth specifies a range of date values that includes all dates from the first to last day of the previous month. So if today's date is September 15, 1999 then LastFullMonth is the same as the range value CDate (#Aug 1, 1999#) To CDate (#Aug 31, 1999#).

Ranges are often used with If or Select expressions. The following example computes student letter grades based on their test scores. Scores greater than or equal to 90 receive an "A", scores from 80 to 90, not including 90, receive a "B" and so on.

```
//Compute student letter grades
Select {Student.Test Scores}
   Case UpFrom 90 :
      "A"
   Case 80 To_ 90 :
      "B"
   Case 70 To_ 80 :
      "C"
   Case 60 To_ 70 :
      "D"
   Default :
      "F";
```

The above example uses the Select expression which is discussed in more detail in Control Structures (Crystal Syntax). You can check if a value is in a range by using the In operator. For example:

```
5 In 2 To 10; //True
```

```
5 In 2 To_ 5; //False
```

```
5 In 2 To 5; //True
```

The Maximum and Minimum functions can be used to find the endpoints of a range:

```
Maximum (2 To 10) //Returns 10
```

Array Data Types (Crystal Syntax)

Arrays in Crystal Reports are ordered lists of values that are all of the same type. These values are known as the array's elements. The elements of an array can be any simple type or range type. Arrays can be created using square brackets ([ ]).

Arrays are most useful when used with variables. Using variables, you can change the individual elements of an array and resize the array to accommodate more elements. For

example, you can accumulate database field values into a global array variable in a detail level formula, and then use a formula in a group footer to perform a calculation based on those values. This enables you to perform a wide variety of customized summary operations.

Examples

An array of three Number values. The first element is 10, the second is 5, and the third is 20.

```
[10, 5, 20]
```

An array of seven String values:

```
["Sun", "Mon", "Tue", "Wed", "Th", "Fri", "Sat"]
```

An array of two DateTime Range values:

```
[#Jan 1, 1998# To #Jan 31, 1998#,
 #Feb 1, 1999# To #Feb 28, 1999#]
```

You can extract individual elements out of an array using square brackets containing the index of the element you want. This is called subscripting the array:

```
[10, 5, 20] [2] //Equal to 5
```

Number ranges can also be used to subscript arrays. The result is another array. For example:

```
[10, 5, 20] [2 To 3] //Equal to [5, 20]
```

Variables (Crystal Syntax)

A variable represents a specific data item, or value, and acts as a placeholder for that value. When a formula encounters a variable, the formula searches for the value of the variable and uses it in the formula. Unlike a constant value, which is fixed and unchanging, a variable can be repeatedly assigned different values. You assign a value to a variable and the variable maintains the value until you later assign a new value. Because of this flexibility, it is necessary for you to declare variables before you use them so that Crystal Reports is aware of them and understands how you intend to use them.

Example

If you wanted to report on customers by area code, you could create a variable that extracts the area code from a customer fax number. The following is an example of a variable called areaCode:

```
Local StringVar areaCode;

areaCode := {Customer.Fax} [1 To 3];
```

The first line of the variable example is the variable declaration; it gives the variable a name and type. The database field {Customer.Fax} is a String field and [1 To 3] extracts the first three characters from its current value. The variable areaCode is then assigned this value.

Variable Declarations (Crystal Syntax)

Before using a variable in a formula, you must declare it. A variable can hold values of a given type. The allowed types are the seven simple types (Number, Currency, String,

Boolean, Date, Time and DateTime), the six range types (Number Range, Currency Range, String Range, Date Range, Time Range and DateTime Range) and variables that hold arrays of the previously mentioned types. This gives a total of 26 different types that a variable can have.

When you declare a variable, you also specify its name. A variable cannot have the same name as any function, operator or other keyword that is valid for Crystal syntax. For example, your variable cannot be named Sin, Mod or If because Sin is a built in function, Mod is a built in operator and If is a built in keyword. When typing formulas in the formula editor, the names of the built in functions, operators and other keywords are highlighted in a different color and so it is easy to check if the variable name conflicts.

Once a variable is declared, it can be used in the formula. For example, you might want to assign it an initial value:

```
Local NumberVar x; //Declare x to be a Number variable
x := 10; //Assign the value of 10 to x
```

> **Note** The keyword for declaring the Number variable has a Var at the end. This is true for all the variable types in Crystal syntax.

A variable can only hold values of one type. For example, if a variable holds a Number value, you cannot later use it to hold a String.

Example

```
Local StringVar y;
y := "hello";
//OK- the Length function expects a String argument
Length (y);
//Error- y can only hold String values
y := #Jan 5, 1993#;
//Error- y can only hold String values
y := ["a", "bb", "ccc"];
//Error- the Sin function expects a Number argument
Sin (y);
```

You can combine declaring a variable and assigning it a value in a single expression. For example:

```
Local NumberVar x := 10 + 20;
Local StringVar y := "Hello" + " " + "World";
Local DateVar z := CDate (#Sept 20, 1999#);
Local NumberVar Range gradeA := 90 To 100;
```

This is a good practice because it is more efficient and helps prevent the common mistake of having incorrectly initialized variables.

Here are some more examples of declaring and initializing range variables:

```
Local NumberVar Range gradeA;
Local DateVar Range quarter;
```

```
gradeA := 90 To 100;
quarter := CDate (1999, 10, 1) To CDate (1999, 12, 31);
```

Variable Scope (Crystal Syntax)

Variable scopes are used to define the degree to which variables in one formula are made available to other formulas. There are three levels of scope in Crystal Reports:

- Local (Crystal Syntax)
- Global (Crystal Syntax)
- Shared (Crystal Syntax)

Every variable has a scope, and this scope is specified when the variable is declared.

Local Variables (Crystal Syntax)

Variables with local scope are declared using the Local keyword followed by the type name (with the Var suffix) followed by the variable name.

Local variables are restricted to a single formula and a single evaluation of that formula. This means that you cannot access the value of a local variable in one formula from a different formula.

Example

```
//Formula A
Local NumberVar x;
x := 10;
//Formula B
EvaluateAfter ({@Formula A})
Local NumberVar x;
x := x + 1;
```

The function call EvaluateAfter ({@Formula A}) ensures that Formula B will be evaluated after Formula A is evaluated. Formula A returns a value of 10 and Formula B returns a value of 1. Formula B does not have access to Formula A's x and thus cannot use the value of 10 and add 1; instead, it uses the default value for the uninitialized local variable x found in Formula B, which is 0, and adds 1 to it to get 1.

You can also create local variables with the same name but different types in different formulas. For example, the type declarations in formulas A and B do not conflict with:

```
//Formula C
Local StringVar x := "hello";
```

Local variables are the most efficient of the three scopes. In addition, they do not interfere with one another in different formulas. For these reasons, it is best to declare variables to be local whenever possible.

Global Variables (Crystal Syntax)

Global variables use the same memory block to store a value throughout the main report. This value is then available to all formulas that declare the variable, except for those in subreports. You declare a global variable as in the following example:

```
Global StringVar y;
```

You can also omit the Global keyword which creates a Global variable by default:

```
StringVar y; //Same as: Global StringVar y;
```

However, even though global variables are easy to declare, it is recommended that you use them only when local variables do not suffice.

Since global variables share their values throughout the main report, you cannot declare a global variable in one formula with one type and then declare a global variable with the same name in a different formula with a different type.

Example

```
//Formula A

Global DateVar z;

z := CDate (1999, 9, 18)

//Formula B

NumberVar z;

z := 20
```

In this case, if you enter and save Formula A first, Crystal Reports will return an error when you check or try to save Formula B. This is because the declaration of the Global variable z as a Number conflicts with its earlier declaration in Formula A as a Date.

When to Use Global Variables

Global variables are often used to perform complex calculations where the results of a formula depend upon the grouping and page layout of the actual printed report. This is accomplished by creating several formulas, placing them in different sections of the report, and having the different formulas interact via global variables.

Example

```
//Formula C

Global NumberVar x;

x := 10;

//Formula D

//Call the function WhileReadingRecords

WhileReadingRecords;

Global NumberVar x;

x := x + 1
```

If Formula C is placed in the Report Header and then Formula D is placed in a detail section, Formula C will be evaluated before Formula D. Formula C will be evaluated once

and then Formula D will be evaluated for each record appearing in the detail section. Formula C returns 10. For the first detail record, Formula D returns 11. This is because the value 10 of x is retained from when it was set by Formula C. Formula D then adds 1 to this value, setting x to 11 and then returns 11. For the second detail record, formula D return 12, adding 1 to the previously retained value of x which was 11. This process continues for the remaining detail records.

The call to WhileReadingRecords tells Crystal Reports to re-evaluate Formula D as it reads in each record of the report. Otherwise, since the formula does not contain any database fields, the program evaluates it only once before reading the records from the database. The formula will then return the value 11 instead of 11, 12, 13, ... as the successive records are processed.

If the expression x := x + 1 is replaced by x := x + {Orders Detail.Quantity}, you create the effect of a running total based on {Orders Detail.Quantity}, although it is one starting at 10 rather than 0 because of Formula C. In this case, you can omit the call to WhileReadingRecords, since it will automatically occur because the formula contains a database field.

Shared Variables (Crystal Syntax)

Shared variables use the same memory block to store the value of a variable throughout the main report and all of its subreports. Thus shared variables are even more general than global variables. To use a shared variable, declare it in a formula in the main report as in the following example:

```
Shared NumberVar x := 1000;
```

and declare it in a formula in the subreport as in the following example:

```
Shared NumberVar x;
```

In order to use shared variables, the variable must be declared and assigned a value before it can be passed between the main report and the subreport.

Declaring Array Variables (Crystal Syntax)

You can declare array variables by following the type name with the keyword Array.

Example

```
//Declare x to be a Global variable of
//Number Array type
Global NumberVar Array x := [10 , 20, 30];
//cost is a Global variable of Currency Array type
//It is automatically Global since the scope specifier
//(one of Local, Global or Shared) is omitted.
CurrencyVar Array cost := [$19.95, $79.50, $110.00,
                           $44.79, $223.99];
//payDays is a Global variable of Date Array type
Global DateVar Array payDays := [CDate(1999, 5, 15),
```

```
CDate(1999, 5, 31)];
//y is a Shared variable of String Range Array type
Shared StringVar Range Array y := ["A" To "C",
                                   "H" To "J"];
//days is a Local variable of String Array type
Local StringVar Array days;
days := ["Sun", "Mon", "Tue", "Wed", "Th",
         "Fri", "Sat"];
```

Assigning Values to Elements of an Array

You can assign values to elements of an array and also use the values of the elements for other computations.

Example

```
StringVar Array x := ["hello", "bye", "again"];
x [2] := "once"; //Now x is ["hello", "once", "again"]
//The expression below would cause an error if not
//commented out since the array has size 3
//x [4] := "zap";
//The formula returns the String "HELLO"
UpperCase (x [1])
```

The Redim and Redim Preserve keywords can be used to resize an array if you want to add extra information to it. Redim erases the previous contents of the array first before resizing it whereas Redim Preserve preserves the previous contents.

```
Local NumberVar Array x;
Redim x [2]; //Now x is [0, 0]
x [2] := 20; //Now x is [0, 20]
Redim x [3]; //Now x is [0, 0, 0]
x [3] := 30; //Now x is [0, 0, 30]
Redim Preserve x [4]; //Now x is [0, 0, 30, 0]
"finished"
Local StringVar Array a;
Redim a [2];
//Assign a value to the first element of the array a
a[1] := "good";
a[2] := "bye";
//The & operator can be used to concatenate strings
a[1] & a[2] //The formula returns the String "goodbye"
```

Arrays and For Loops

Arrays are commonly used with For loops. The following example creates and then uses the Array [10, 20, 30, ..., 100] using a For loop.

```
Local NumberVar Array b;
Redim b[10];
Local NumberVar i;
For i := 1 To 10 Do
(
   b[i] := 10 * i
);
b [2] //The formula returns the Number 20
```

Default Values for Simple Types (Crystal Syntax)

An uninitialized variable will have the default value for its type. In general, it is not a good programming practice to rely on the default values of types. For example, initialize all local variables in your formula, initialize all global variables in a formula placed in the Report Header and initialize all shared variables in a formula placed in the Report Header of the main report.

When an array is resized using the Redim keyword, the entries are filled with default values for the type.

Default values

Number

```
0
```

Currency

```
$0
```

String

```
"" //The empty string
```

Date

```
Date (0, 0, 0) //The null Date value
```

Time

The null Time value. Value held by an uninitialized Time variable.

DateTime

The null DateTime value. Value held by an uninitialized DateTime variable.

Note It is not recommended that your formulas rely on the values of uninitialized range or array variables.

Automatic Type Conversions (Crystal Syntax)

Generally in Crystal Reports, values of one type cannot be used where values of another type are expected without explicitly supplying a type conversion function. For example:

```
Local StringVar postalCode;
//Error- assigning a Number value to a String
postalCode := 10025;
//OK - use the type conversion function CStr
//to create "10025"
postalCode := CStr (10025, 0);
```

However, there are a few conversions that are made automatically:

Number to Currency

Date to DateTime

Simple type to Range value of the same underlying simple type

For example, the following assignments are correct:

```
Local CurrencyVar cost;
//Same as: cost := $10
cost := 10;
Local DateTimeVar orderDate;
//Same as: orderDate := CDateTime (1999, 9, 23, 0, 0, 0)
orderDate := CDate (1999, 9, 23);
Local NumberVar Range aRange;
//Same as: aRange := 20 To 20
aRange := 20;
Local NumberVar Range Array aRangeArray;
//Same as : aRangeArray := [10 To 10, 20 To 25, 2 To 2]
aRangeArray := [10, 20 To 25, 2];
```

Note The opposite conversions are not allowed. For example:

```
Local NumberVar num;
num := 5 + $10; //Error
//OK- convert to Number type using the CDbl function
num := CDbl (5 + $10) //Could also use ToNumber
```

5 is converted to $5 and added to $10 to make $15. However, this Currency value cannot be automatically assigned to the Number variable num since automatic conversions from Currency to Number are not allowed. Similarly, functions accepting a Currency argument can be supplied a Number argument instead, and the Number argument will be converted to a Currency, whereas functions accepting a Number argument cannot be supplied a

Currency argument without first explicitly converting the Currency to a Number using CDbl.

Functions (Crystal Syntax)

Functions are built-in procedures or subroutines used to evaluate, make calculations on, or transform data. When you specify a function, the program performs the set of operations built into the function without you having to specify each operation separately.

Functions Overview (Crystal Syntax)

When using a function in a formula, type the name of the function and supply the arguments required. For example, the Length function requires a String argument and computes the length of the string.

```
Local StringVar x := "hello";

Length (x) //The formula returns the Number 5
```

Supplying arguments of the incorrect type required by the function produces an error. For example, calling Length (3) would produce an error since Length does not accept a Number argument.

Functions sometimes can accept different numbers of arguments or types of arguments. For example, the CDate function which could accept a single String argument to form a Date value or three Number values holding the year, month and day respectively and form a Date value from them.

Example with the Mid function

```
Local StringVar x := "hello";

Local StringVar y;

//Start at position 2, go to the end of the string

y := Mid (x, 2); //y is now "ello"

//Start at position 2, extract 1 character

y := Mid (x, 2, 1) //y is now "e"
```

These classes of functions are: Math, Summary, Financial, String, Date/Time, Date Range, Array, Type Conversion, Programming Shortcuts, Evaluation Time, Print State, Document Properties and Additional Functions. There are also some functions specific to conditional formatting formulas.

Non Reporting-Specific Functions (Crystal Syntax)

The Math, Financial, String, Date/Time, Type Conversion, and Programming Shortcuts groups consist mainly of functions that are not specific to reporting, but might be found in any full featured programming environment. Many of the functions are similar in functionality to Visual Basic functions of the same or similar name.

> **Note** Some functions that are supported by Crystal syntax are not listed in the Crystal syntax Functions tree. This is because they are equivalent to Crystal syntax functions already listed in the tree.

For example, the Length function is the traditional Crystal syntax function for finding the length of a string. Crystal syntax also supports Len as a synonym. Len is the Visual Basic and Basic syntax function for this and its inclusion is for the convenience of Visual Basic and Basic syntax users who want to write or modify Crystal syntax formulas.

Conditional Formatting Functions (Crystal Syntax)

When writing a conditional formatting formula, certain additional functions appear at the top of the Functions tree to help you with this. For example, you can format the {Customer.Last Year's Sales} field to print sales of more than $100,000 in green, sales of less than $15,000 in red, and all other sales in black.

Example

```
//Conditional formatting example 1
If {Customer.Last Year's Sales} > 100000 Then
   crGreen
Else If {Customer.Last Year's Sales} < 15000 Then
   crRed
Else
   crBlack
```

Since this is a font color formatting function, the list of Color Constants appears in the Functions Tree. This example uses three: crGreen, crRed and crBlack. You could have used the actual numeric values of the color constants instead. For example, crRed is 255 and crGreen is 32768. However, the formula is more understandable using the color constants. All constant functions in Crystal syntax can have the "cr" prefix.

Crystal syntax still supports constant functions from previous versions without the "cr" prefix. For example, you can use "Red" instead of "crRed". However, using the "cr" prefix organizes constant functions and is recommended.

> **Note** Some formatting attributes do not use constant functions. For example, if you wanted to not print {Customer.Last Year's Sales} values if the sales were less than $50,000, you could write the following conditional formatting formula for the suppress attribute:

```
//Conditional formatting example 2
If {Customer.Last Year's Sales} < 50000 Then
   True //suppress the value
Else
   False //do not suppress the value
```

Or more simply:

```
//Conditional formatting example 3 -
//equivalent to example 2
{Customer.Last Year's Sales} < 50000
```

If the last year's sales are less than $50,000, then the expression

```
{Customer.Last Year's Sales} < 50000
```

is True, and so the formula returns True. On the other hand, if the last year's sales are greater than or equal to $50,000, then

```
{Customer.Last Year's Sales} < 50000
```

is False and so the formula returns False.

General Purpose Conditional Formatting Functions (Crystal Syntax)

There are three general purpose conditional formatting functions:

CurrentFieldValue

DefaultAttribute

GridRowColumnValue

DefaultAttribute can be used for any formatting formula, **CurrentFieldValue** for any formatting formula where you are formatting a field value, and **GridRowColumnValue** for any formatting formula where you are formatting a field value in a Cross-Tab or OLAP grid.

CurrentFieldValue enables you to conditionally format Cross-Tab or OLAP grid cells based on their value. **GridRowColumnValue** enables you to conditionally format the cells of a Cross-Tab or OLAP grid based on row or column headings values. These two functions are essential in some situations as there is no other way in the formula language to refer to these fields.

Example

If you wanted Cross-Tab cells to be suppressed if the values are less than 50,000:

```
//Conditional formatting example 4

CurrentFieldValue < 50000
```

Operators (Crystal Syntax)

Operators are special symbols or words that describe an operation or an action to take place between two or more values. The program reads the operators in a formula and performs the actions specified.

Arithmetic Operators (Crystal Syntax)

Comparison Operators (Crystal Syntax)

Boolean Operators (Crystal Syntax)

Null Fields and Null Values (Crystal Syntax)

Arithmetic Operators (Crystal Syntax)

Arithmetic operators are used to combine numbers, numeric variables, numeric fields and numeric functions to get another number.

The arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), integer division (\), percent (%), modulus (Mod), negation (-) and exponentiation (^).

Examples

```
//Outstanding preferred stock as a percent of
//common stock
{Financials.Preferred Stock} %
{Financials.Common Stock};
//The square root of 9, Sqr(9) is 3
//The formula returns 17
7 + 2 * 3 - 2 + Sqr(6 + 3) * Length("up");
```

Order of Precedence

In general, the program evaluates expressions in the following order:

from left to right

follows the rules of precedence from basic math

List of arithmetic operators, from highest precedence to lowest

Exponentiation (^)

Negation (-)

Multiplication, division, and percent (*, /, %)

Integer Division (\)

Modulus (Mod)

Addition and subtraction (+, -)

Example

Multiplication and division are performed first from left to right. Then addition and subtraction are performed. For example, 5 + 10 * 3 = 5 + 30 = 35.

You can change this order of precedence by using parentheses. For example, (5 + 10) * 3 = 15 * 3 = 45. If you are unsure of the order of precedence, it is a good idea to clarify your intentions with parentheses.

Comparison Operators (Crystal Syntax)

Comparison operators are usually used to compare operands for a condition in a control structure such as an If expression.

The comparison operators are equal (=), not equal (<>), less than (<), less than or equal (<=), greater than (>) and greater than or equal (>=).

Comparison operators as a group all have lower precedence than the arithmetic operators. Thus, expressions like 2 + 3 < 2 * 9 are the same as (2 + 3) < (2*9).

Boolean Operators (Crystal Syntax)

Boolean operators are typically used with comparison operators to generate conditions for control structures.

The Boolean operators are, in order of precedence from greatest to lowest: Not, And, Or, Xor, Eqv and Imp.

Boolean operators as a group have lower precedence than the comparison operators. Thus for example, the expression 2 < 3 And 4 >= -1 is the same as (2 < 3) And (4 >= -1).

Null Fields and Null Values (Crystal Syntax)

In general, when Crystal Reports encounters a null valued field in a formula, it immediately stops evaluating the formula and produces no value. If you want to handle null field values in your formula, you must explicitly do so using one of the special functions designed for handling them: IsNull, PreviousIsNull or NextIsNull.

Relating to operators, when Crystal Reports evaluates the condition:

```
IsNull({Product.Color}) Or
InStr({Product.Color}, " ") = 0
```

It first evaluates IsNull ({Product.Color}), and when it determines that this is True, it knows that the whole condition is True, and thus does not need to check whether

```
InStr({Product.Color}, " ") = 0
```

In other words, Crystal Reports will stop evaluating a Boolean expression when it can deduce the results of the whole expression. In the following example, the formula guards against attempting to divide by zero in the case that denom is 0:

```
Local NumberVar num;
Local NumberVar denom;
...
If denom <> 0 And num / denom > 5 Then
...
```

Example

The {Product.Color} field contains both basic colors such as "red" and "black" and fancy two word colors such as "steel satin" and "jewel green". Suppose that you want to write a formula that writes out "basic" for the basic colors and "fancy" for the others.

```
If InStr({Product.Color}, " ") = 0 Then
   "basic"
Else
   "fancy"
```

The function call to InStr searches the {Product.Color} string for a space. If it finds a space, it returns the position of the space, otherwise it returns 0. Since basic colors are only one word with no spaces, InStr will return 0 for them.

For some products, such as the Guardian Chain Lock, a color value was not recorded and so the {Product.Color} field has a null value in the database for that record. Thus, the Guardian Chain Lock record does not have any word printed beside it.

Here is how to fix up the previous example using IsNull:

```
If IsNull({Product.Color}) Or
   InStr({Product.Color}, " ") = 0 Then
   "basic"
```

```
Else
   "fancy"
```

Control Structures (Crystal Syntax)

Formulas without control structures execute each expression in the formula exactly once when the formula is evaluated. The expressions are executed in a sequential fashion, from the first expression in the formula to the last. Control structures enable you to vary this rigid sequence. Depending upon which control structure you choose, you can skip over some of the expressions or repeatedly evaluate some expressions depending on certain conditions. Control structures are the primary means of expressing business logic and typical report formulas make extensive use of them.

If Expressions (Crystal Syntax)

The If expression is one of the most useful control structures. It allows you to evaluate an expression if a condition is true and evaluate a different expression otherwise.

> **Note** When formatting with conditional formulas, always include the Else keyword; otherwise, values that don't meet the If condition may not retain their original format. To prevent this, use the DefaultAttribute function (If...Else DefaultAttribute).

Example

A company plans to pay a bonus of 4 percent to its employees except for those who work in Sales who will receive 6 percent. The following formula using an If expression would accomplish this:

```
//If example 1
If {Employee.Dept} = "Sales" Then
   {Employee.Salary} * 0.06
Else
   {Employee.Salary} * 0.04
```

In this example, if the condition {Employee.Dept} = "Sales" evaluates as true, then the

```
{Employee.Salary} * 0.06
```

expression is processed. Otherwise the expression following the Else, namely the

```
{Employee.Salary} * 0.04
```

is processed.

Suppose another company wants to give employees a 4% bonus, but with a minimum bonus of $1,000. Notice that the Else clause is not included; it is optional, and not needed in this case.

```
//If example 2
Local CurrencyVar bonus := {Employee.Salary} * 0.04;
If bonus < 1000 Then
   bonus := 1000;
//The final expression is just the variable 'bonus'.
```

```
//This returns the value of the variable and is the
//result of the formula
bonus
```

Another way of accomplishing example 2 is to use an Else clause:

```
//If example 3
Local CurrencyVar bonus := {Employee.Salary} * 0.04;
If bonus < 1000 Then
   1000
Else
   bonus
```

Now suppose that the previous company also wants a maximum bonus of $5,000. You now need to use an Else If clause. The following example has only one Else If clause, but you can add as many as you need.

Note There is a maximum of one Else clause per If expression.

The Else clause is executed if none of the If or Else If conditions are true.

```
//If example 4
Local CurrencyVar bonus := {Employee.Salary} * 0.04;
If bonus < 1000 Then
   1000
Else If bonus > 5000 Then
   5000
Else
   bonus;
```

Example

Suppose that a company wants to compute an estimate of the amount of tax an employee needs to pay and write a suitable message. Income below $8,000 is not taxed, income between $8,000 to $20,000 is taxed at 20%, income between $20,000 to $35,000 is taxed at 29%, and income above $35,000 is taxed at 40%.

```
//If example 5
Local CurrencyVar tax := 0;
Local CurrencyVar income := {Employee.Salary};
Local StringVar message := "";
If income < 8000 Then
(
   message := "no";
   tax := 0
)
```

```
Else If income >= 8000 And income < 20000 Then
(
   message := "lowest";
   tax := (income - 8000)*0.20
)
Else If income >= 20000 And income < 35000 Then
(
   message := "middle";
   tax := (20000 - 8000)*0.20 + (income - 20000)*0.29
)
Else
(
   message := "highest";
   tax := (20000 - 8000)*0.20 + (35000 - 20000)*0.29 +
          (income - 35000)*0.40
);
//Use 2 decimal places and the comma as a
//thousands separator
Local StringVar taxStr := CStr (tax, 2, ",");
"You are in the " & message & " tax bracket. " &
"Your estimated tax is " & taxStr & "."
```

Note The use of variables is to simplify the logic of the computation. Also, there are two expressions that are executed when one of the conditions are met; one assigns the tax variable, and the other assigns the message variable. It is often useful to have multiple expressions executed as a result of a condition.

Grouping Expressions (Crystal Syntax)

The If expression is an expression. In other words it evaluates to a value of a given type. If there is no Else clause, and the condition is not true, then the value is the default value for the type. For example:

```
If Length ({Employee.First Name}) < 5 Then
   "short"
```

The above If expression returns a String value. The string value is "short" if the Employee's first name has fewer than 5 letters and the empty String "" otherwise.

Consider the formula:

```
If Year({Orders.Order Date}) >= 1995 Then
   {Orders.Order Date}
```

For order dates before 1995, the above If expression returns the null DateTime value. It is a DateTime value rather than a Date value since {Orders.Order Date} is a DateTime database field. The null DateTime value is not printed by Crystal Reports so if the above formula is placed in a report, the formula field would be blank for order dates before 1995. Null Time values and null Date values behave similarly.

Here is an example that illustrates the use of parentheses to have more than one expression executed as the outcome of an If condition. A company charges a 5 percent fee for orders shipped within three days and a 2 percent fee otherwise. It wants to print messages such as "Rush shipping is $100.00" or "Regular shipping is $20.00" as appropriate.

```
Local StringVar message;
Local CurrencyVar ship;
If {Orders.Ship Date} - {Orders.Order Date} <= 3 Then
(
   message := "Rush";
   //A semicolon at the end of the next line
   //is optional
   ship := {Orders.Order Amount} * 0.05
) //A semicolon cannot be placed here
Else
(
   message := "Regular";
   ship := {Orders.Order Amount} * 0.02;
);
//The preceding semicolon is required to separate the
//If expression from the final expression below
message & " shipping is " & CStr (ship)
```

When expressions are grouped together with parentheses, the whole group is considered as a single expression, and its value and type are the value and type of the final expression inside the parentheses.

```
//The parentheses group expression as a whole has
//Currency type
(
   //The first expression in the parentheses has
   //String type
   message := "Rush";
   //The second and final expression in parentheses
   //has Currency type
```

```
   ship := {Orders.Order Amount} * 0.05;
)
```

Thus, for example, the following formula gives an error. The reason is that the Then part of the If expression returns a Currency value while the Else part returns a String value. This is not allowed, since the If expression is an expression and so must always return a value of a single type.

```
//An erroneous formula
Local StringVar message;
Local CurrencyVar ship;
If {Orders.Ship Date} - {Orders.Order Date} <= 3 Then
(
   message := "Rush";
   ship := {Orders.Order Amount} * 0.05
)
Else
(
   //The following 2 lines were interchanged
   ship := {Orders.Order Amount} * 0.02;
   message := "Regular";
);
message & " shipping is " & CStr (ship)
```

One way of fixing up the erroneous formula without being careful about expression order is just to make the If expression return a constant value of the same type in every branch. For example, the If expression now returns the Number value 0:

```
//Repaired the erroneous formula
Local StringVar message;
Local CurrencyVar ship;
If {Orders.Ship Date} - {Orders.Order Date} <= 3 Then
(
   message := "Rush";
   ship := {Orders.Order Amount} * 0.05;
   0
)
Else
(
   ship := {Orders.Order Amount} * 0.02;
   message := "Regular";
```

```
   0
);
message & " shipping is " & CStr (ship)
```

Select Expressions (Crystal Syntax)

The Select expression is similar to an If expression. Sometimes however, you can write clearer and less repetitive formulas using the Select expression. This example evaluates the {Customer.Fax} field to determine if the area code is for Washington state (206, 360, 509) or British Columbia, Canada (604, 250):

```
//Select example 1
Select {Customer.Fax}[1 To 3]
   Case "604", "250" :
      "BC"
   Case "206", "509", "360" :
      "WA"
   Default :
      "";
```

The expression right after the Select keyword is called the Select condition. In the above example it is {Customer.Fax}[1 To 3]. The Select expression tries to find the first Case that matches the Select condition, and then executes the expression following the colon for that Case. The Default case is matched if none of the preceding cases match the Select condition. Notice that there is also a colon after the Default.

```
//Same effect as Select example 1
Local StringVar areaCode := {Customer.Fax}[1 To 3];
If areaCode In ["604", "250"] Then
   "BC"
Else If areaCode In ["206", "509", "360"] Then
   "WA"
Else
   "";
```

Example

This formula groups the number of Oscar nominations a movie received into low, medium, high or extreme categories and in the process, shows some of the possibilities for the expression lists following the Case labels.

```
//Select example 2
Select {movie.NOM}
   Case 1,2,3, Is < 1 :
   (
```

```
      //Can have expression lists by using
      //parentheses
      10 + 20;
      "low"
   )
   Case 4 To 6, 7, 8, 9 :
      "medium"
   Case 10 :
      "high"
   Default :
      "extreme"
```

The Default clause of the Select expression is optional. If the Default clause is missing and none of the cases are matched, then the Select expression returns the default value for its expression type. For example, if in the above example the Default clause were omitted and {movie.NOM} = 11, it would return the empty string "". The Select expression is an expression, and similar comments as in the More details on If expressions section apply to it as well.

For Loops (Crystal Syntax)

For loops enable you to evaluate a sequence of expressions multiple numbers of times. This is unlike the If and Select expressions where the program passes through each expression at most once during the formula's evaluation.

For loops are best when you know the number of times that the expressions needs to be evaluated in advance.

For Loop Syntax

Example 1

Suppose you want to reverse the {Customer.Customer Name} string. For example, "City Cyclists" becomes "stsilcyC ytiC".

```
//Reverse a string version 1
Local StringVar str := "";
Local NumberVar strLen :=
   Length ({Customer.Customer Name});
Local NumberVar i;
For i := 1 To strLen Do
(
   Local NumberVar charPos := strLen - i + 1;
   str := str + {Customer.Customer Name}[charPos]
);
```

```
str
```

Examine how this formula works assuming that the current value of the field {Customer.Customer Name} is "Clean Air". The variable strLen is assigned to be the length of "Clean Air", namely 9. The variable i is known as a For counter variable since its value changes with each iteration of the For loop. In other words, it is used to count the iterations of the loop. The For loop will iterate 9 times, during the first time, i is 1, then i is 2, then i is 3 and so on until finally i = 9. During the first iteration, the ninth character of {Customer.Customer Name} is appended to the empty string variable str. Thus str equals "r" after the first iteration. During the second iteration, the eighth character of {Customer.Customer Name} is appended to str and so str equals "ri". This continues until after the ninth iteration, str equals, "riA naelC" which is the reversed string.

Example 2

Here is a simpler version of the above formula that uses a Step clause with a negative Step value of -1. For the "Clean Air" example, i is 9 for the first iteration, 8 for the second, 7 for the third and so on until it is 1 in the final iteration.

```
//Reverse a string version 2

Local StringVar str := "";

Local NumberVar strLen :=

   Length ({Customer.Customer Name});

Local NumberVar i;

For i := strLen To 1 Step -1 Do

(

   str := str + {Customer.Customer Name}[i]

);

str
```

Example 3

The simplest version is to use the built in function StrReverse:

```
//Reverse a string version 3

StrReverse ({Customer.Customer Name})
```

The built in String functions in Crystal Reports can handle many of the string processing applications that would traditionally be handled using a For loop or some other kind of loop. However, For loops provide the most flexibility in processing strings and also power in processing arrays, which can be essential if the built-in functions do not cover your intended application.

For Loop Example (Crystal Syntax)

Here is a more detailed example of Crystal Reports' string processing capabilities. The Caesar cipher is a simple code that is traditionally credited to Julius Caesar. In this code, each letter of a word is replaced by a letter five characters further in the alphabet. For example, "Jaws" becomes "Ofbx". Notice that "w" is replaced by "b"; since there are not 5 characters after "w" in the alphabet, it starts again from the beginning.

Here is a formula that implements applying the Caesar cipher to the field {Customer.Customer Name} in the Xtreme database:

```
//The Caesar cipher
//The input string to encrypt
Local StringVar inString := {Customer.Customer Name};
Local NumberVar shift := 5;
Local StringVar outString := "";
Local NumberVar i;
For i := 1 To Length(inString) Do
(
   Local StringVar inC := inString [i];
   Local StringVar outC;
   Local BooleanVar isChar :=
   LowerCase(inC) In "a" To "z";
   Local BooleanVar isUCaseChar :=
   isChar And (UpperCase (inC) = inC);
   inC := LCase(inC);
   If isChar Then
   (
      Local NumberVar offset :=
         (Asc(inC) + shift - Asc("a")) Mod
         (Asc("z") - Asc("a") + 1);
      outC := Chr(offset + Asc("a"));
      If isUCaseChar Then outC := UpperCase(outC)
   )
   Else
      outC := inC;
   outString := outString + outC
);
outString
```

In the above example there is an If expression nested within the expression block of the For loop. This If expression is responsible for the precise details of shifting a single character. For example, letters are treated differently from punctuation and spaces. In particular, punctuation and spaces are not encoded. The general points here are that control structures can be nested within other control structures and that multiple expressions can be included in the (parentheses enclosed) expression blocks of other control structures.

Exiting from For Loops (Crystal Syntax)

You can exit from a For loop by using Exit For. The following example searches the Global array names for the name "Fred". If it finds the name, it returns the index of the name in the array. Otherwise it returns -1.

For example, if the names array is:

```
["Frank", "Helen", "Fred", "Linda"]
```

Then the formula returns 3.

```
Global StringVar Array names;
//The names array has been initialized and filled
//in other formulas
Local NumberVar i;
Local NumberVar result := -1;
//The UBound function returns the size of its array
//argument
For i := 1 to UBound (names) Do
(
   If names [i] = "Fred" Then
   (
      result := i;
      Exit For
   )
);
result
```

When considered as an expression, the For loop always returns the Boolean value True. Thus you will almost never want a For loop to be the last expression in a formula, since then the formula will then just display the value True rather than your intended result.

While Loops (Crystal Syntax)

A While loop can be used to execute a fixed block of statement an indefinite amount of time.

Two Types of While Loops

Type of Loop	Explanation	Example
While ... Do	The While ... Do loop evaluates the condition, and if the condition is true, then it evaluates the expression following the Do.	`While condition Do` `  expression`

	When it has finished doing this, it evaluates the condition again and if the condition is true, it evaluates the expression following the Do again. It continues repeating this process until the condition is false.	

Do ... While	The Do ... While loop evaluates the expression once no matter what.
	It then evaluates the condition, and if the condition is true, evaluates the expression again. This process continues until the condition is false.
		`Do`
		`  expression`
		`While condition`

Note The While loops support an Exit While statement to immediately jump out of the loop. Its use is analogous to the use of Exit For in For loops. As with the For loop, the While loop when considered as an expression always returns the Boolean value True.

While ... Do loop Example

The following example searches for the first occurrence of a digit in an input string. If a digit is found, it returns its position, otherwise it returns -1. In this case, the input string is set explicitly to a string constant, but it could be set equal to a String type database field instead.

For example, for the input String, "The 7 Dwarves", the formula returns 5, which is the position of the digit 7.

```
Local StringVar inString := "The 7 Dwarves";
Local NumberVar strLen := Length (inString);
Local NumberVar result := -1;
Local NumberVar i := 1;
While i <= strLen And result = -1 Do
(
   Local StringVar c := inString [i];
   If NumericText (c) Then
      result := i;
   i := i + 1;
);
result
```

Preventing Infinite Loops (Crystal Syntax)

There is a safety mechanism to prevent report processing from hanging due to an infinite loop. Any one evaluation of a formula can have at most 100,000 loop condition evaluations per formula evaluation. For example:

```
Local NumberVar i := 1;
While i <= 200000 Do
(
   If i > {movie.STARS} Then
      Exit While;
   i := i + 1
);
20
```

If {movie.STARS} is greater than 100,000 then the loop condition (i <= 200000) will be evaluated more than the maximum number of times and an error message is displayed. Otherwise the loop is OK.

Note The safety mechanism applies on a per formula base, not for each individual loop. For example:

```
Local NumberVar i := 1;
For i := 1 To 40000 Do
(
   Sin (i);
);
While i <= 70000 Do
(
   i := i + 1;
)
```

The above formula also triggers the safety mechanism since the 100,000 refers to the total number of loop condition evaluations in the formula and this formula will have 40001 + 70001 such evaluations.

Formula Size Restrictions

For reference purposes, here are the size restrictions of the formula language:

The maximum length of a String constant, a String value held by a String variable, a String value returned by a function or a String element of a String array is 65,534 characters.

The maximum size of an array is 1000 elements.

The maximum number of arguments to a function is 1000. (This applies to functions that can have an indefinite number of arguments such as Choose).

The maximum number of loop condition evaluations per evaluation of a formula is 100,000.

Date-time functions modeled on Visual Basic accept dates from year 100 to year 9999. Traditional Crystal Reports functions accept dates from year 1 to year 9999.

System Setup

To develop a Web or Windows application that uses Crystal Reports for Visual Studio 2005, check that the following requirements are met:

What Needs to be Installed?

What Needs to be Verified?

Optional Installation: MSDE

64-Bit Development Configuration

What Needs to be Installed?

You need to install Crystal Reports for Visual Studio 2005 before you can create Web or Windows applications that use Crystal reports.

Click the appropriate link to jump to that section:

Visual Studio Versions

Crystal Reports Versions

Visual Studio Versions

Crystal Reports is available for purchase as a separate application, or as a version that is installed as part of most versions of Microsoft Visual Studio (commonly known as Crystal Reports for Visual Studio).

If you intend to use Crystal Reports for Visual Studio, you must verify that the version of Visual Studio that you have installed ships with Crystal Reports for Visual Studio.

Visual Studio is available in the following versions:

Version	Crystal Reports included?
Visual Studio .NET 2002, Standard (C#, Visual Basic, or C++)	No
Visual Studio .NET 2002, Professional or higher	Yes
Visual Studio .NET 2003, Standard (C#, Visual Basic, or C++)	No
Visual Studio .NET 2003, Professional or higher	Yes
Visual Studio 2005 Express Edition (C#, Visual Basic, C++, SQL, J#, Web Developer)	No
Visual Studio 2005, Professional or higher	Yes

The Crystal Reports Component

By default, the Crystal Reports component is installed as part of your Visual Studio installation. If you must install the Crystal Reports component, relaunch the Visual Studio installer and follow directions to add enterprise development tools.

Crystal Reports Versions

Multiple versions of Crystal Reports are available. Many of the procedures in this document are version-specific. To best use this document to your advantage, determine first which version of Crystal Reports you have installed.

To determine which version you are currently using

1. Go to the **GAC (Global Assembly Cache)** at:

   `C:\WINNT\assembly`

   or

   `C:\Windows\assembly`

2. Look for files in this assembly folder whose prefix begins with CrystalDecisions.
3. Locate the CrystalDecisions.CrystalReports.Engine file.

   Note If you have installed more than one version of Crystal Reports, you have multiple versions of these files in the Global Assembly Cache. The CrystalDecisions.CrystalReports.Engine file is selected, because this file is included with every version of Crystal Reports.
4. Note the **Version** column shown in the window.

   This is the "Assembly version."
5. Locate the highest number for a particular assembly.
6. Right-click the file, select **Properties**, and then click the **Versions** tab within the **Properties** window.
7. Compare the file version against the version number in the chart below.

   Note If you are using Crystal Reports version 9.0 with Microsoft Visual Studio .NET 2003, the maintenance release CR 9.2 (available as the CR 9.2 upgrade CD) must be installed. Crystal Decisions supports only CR 9.2 in Visual Studio .NET 2003.

   If Borland C# is installed after Visual Studio .NET 2003, it updates the assembly versions.

Product	Hot Fix Applied?	Optional Add-Ons	Assembly Version	File Version
Crystal Reports for Visual Studio .NET 2002	no	none	9.1.3300	9.1.9360
Crystal Reports for Visual Studio .NET 2002 (patched)	yes	none	9.1.3300	9.1.9466
Crystal Reports 9	no	RAS 9	9.2.3300	9.2.9466
Crystal Reports 9.2 (maintenance release)	yes	RAS 9	9.2.3300	9.2.9500
Crystal Reports for Visual Studio .NET 2003	no	none	9.1.5000	9.1.9800.0
Crystal Reports for Visual Studio .NET 2003 (patched)	yes	none	9.1.5000	9.1.9800.0
Crystal Reports 10	no	RAS 10	10.0.3300.0	10.0.9500.0
Crystal Reports 11	no	RAS 11	11.0.3300.0	11.0.9500.0
Crystal Reports for Visual Studio 2005	no	none	10.2.3600.0	10.2.51014.0
Crystal Reports for Borland C#	no	none	9.1.5000	9.1.9800.0

What Needs to be Verified?

Verification needed to develop projects using Crystal Reports for Visual Studio .NET

To develop a project that uses Crystal Reports for Visual Studio .NET, verify the following:

Add New Item Dialog Box Includes Crystal Reports

Sample Reports' Directory

Tutorials' Sample Code Directory

Viewers' Virtual Directory

Verification needed to work with tutorials

If you also want to work with the tutorials that are provided in this documentation, you must verify that the following additional items are in place:

Location of Xtreme Sample Database

ODBC DSN Entry for Xtreme Sample Database

Optional verification needed to work with SQL Server/MSDE tutorials

If you want to work with the SQL Server/MSDE-specific tutorials that are provided in this documentation, verify that the following SQL Server/MSDE-specific items are in place:

MSDE Installation with Windows or SQL Server Authentication

Northwind Database Installation

Add New Item Dialog Box Includes Crystal Reports

In an earlier section, you learned which Visual Studio version to install and how to install the Crystal Reports component manually, if you did not install it by default during your Visual Studio installation.

To verify that the Crystal Reports component of Visual Studio is installed, check that Crystal Reports appears in the Add New Item dialog box in Visual Studio.

To verify that Crystal Reports is installed

1. Launch Visual Studio.
2. Create a new Web or Windows project (in any language), or open an existing Web or Windows project.
3. On the **Project** menu, click **Add New Item**.
4. In the **Add New Item** dialog box, scroll down and verify that **Crystal Reports** is one of the available items.

64-Bit Development Configuration

Note This section contains information specific to users with a 64-Bit development machine.

Crystal Reports for Visual Studio 2005 projects can be built on a 32-bit or a 64-bit machine. In addition, the projects' code can be ported to and updated on a 64-bit machine. Itanium (IA64) processors are supported for run-time only.

On a 64-bit machine, the 32-bit Crystal Reports for Visual Studio 2005 IDE will run under WOW64 (the x86 emulator that runs 32-bit applications on a 64-bit machine).

Before you can run an application on a 64-bit machine you must install the Crystal Reports 64 bit configuration application for your 64-bit environment.

If you have installed Visual Studio into the default installation directory, the configuration application can be found in the below directory.

x64

```
C:\Program Files\Microsoft Visual Studio 8\Crystal
Reports\CRRedist\X64\CRRedist2005_x64.msi
```

Optional Installation: MSDE

If you want to work with the SQL Server/MSDE-specific tutorials that are provided in this documentation, install the following SQL Server/MSDE-specific items:

MSDE Installation with Windows or SQL Server Authentication

Northwind Database Installation

Security: Creating a Limited Access Database Account

MSDE Installation with Windows or SQL Server Authentication

Some of the tutorials in this documentation show how to display a Crystal report whose data configuration requires a programmatic logon to a secure SQL database.

These tutorials include the following:

Logging onto a Secure SQL Server Database (CrystalReportViewer Object Model)

Logging onto a Secure SQL Server Database Using SQL Authentication (ReportDocument Object Model

Logging onto a Secure SQL Server Database Using Integrated Security (ReportDocument Object Model)

Reduced-Code Secure Database Logon in a Web Site

Connecting to IDataReader

To work with these tutorials, you need Microsoft SQL Server (or its free version, MSDE) installed and configured with either Windows Authentication (if you wish to use Integrated Security) or SQL Server Authentication (if you wish to use SQL Security).

Note Microsoft recommends Integrated Security as the most secure approach.

If you already have Microsoft SQL Server or MSDE installed with either Windows or SQL Server Authentication, proceed to Northwind Database Installation to verify that the sample database "Northwind" is installed.

If you do not have either server installed, this section shows you how to install MSDE with Windows or SQL Server Authentication.

Note To configure your project to connect to Oracle or other SQL database servers, see the knowledge database at: http://support.businessobjects.com/search/

Alternate versions of MSDE

You now have two alternate ways to install MSDE, depending on whether you are using the MSDE shipped with Visual Studio .NET 2002, or MSDE Release A from the Microsoft web site.

Installing MSDE shipped with Visual Studio .NET 2002

You can install the version of MSDE provided with Visual Studio .NET 2002. This version is located in the following file directory:

C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Samples\Setup\msde\setup\sql2000.msi

However, if you use this version, you must upgrade to the latest service pack for MSDE. Go to http://www.microsoft.com/sql/downloads/.

This version uses the sql2000.msi installer.

Note In this step procedure, you install MSDE in Mixed Mode, allowing you to use either Windows Authentication or SQL Server Authentication.

To install MSDE in Mixed Mode using the sql2000.msi installer

1. Go to the command prompt.

   Note Run this installer from the command prompt, to include a command parameter.
2. Change to the directory that contains the installer file, as shown in the table above.
3. Type the command with the following command parameter:

   ```
   sql2000.msi SECURITYMODE=SQL
   ```
4. After the installer has finished, restart your computer.
5. Once the computer is restarted, go to the command prompt again.
6. At the prompt, type this command to log in to the server:

   ```
   osql -U sa
   ```
7. Press ENTER.
8. When asked for a password, press ENTER (at this point, the password is null).

   A prompt appears with the number 1.
9. Decide on a password and enter it with the following command (include all characters shown). In this example, the password 1234 is used.

   ```
   sp_password null, '1234', 'sa'
   ```
10. Go to the next line and type the following:

    ```
    go
    ```

    That word indicates to osql that the command is complete.
11. Press ENTER to execute the command.

    This message appears:

    ```
    Password changed.
    ```
12. Type `exit`, and then press ENTER.

    Finally, you **must** upgrade to the latest service pack for MSDE.
13. Download the service pack from http://www.microsoft.com/sql/downloads/.
14. Install the service pack.

This completes your installation of MSDE using the sql2000.msi installer. In the next section, you verify that the Northwind database is installed.

Installing MSDE Release A from the Microsoft web site

You can download MSDE Release A from the Microsoft website. Go to http://www.microsoft.com/sql/downloads/.

This version uses a setup.exe installer.

Note In this step procedure, you install MSDE in Mixed Mode, allowing you to use either Windows Authentication or SQL Server Authentication.

To install MSDE Release A in Mixed Mode using the setup.exe installer

1. Unzip the MSDE Release A download zip file to a file directory on your hard drive.
2. Launch the command prompt.

Note Run this installer from the command prompt, to include a command parameter.

3. Change to the file directory that you have just unzipped.

   Note This directory contains the setup.exe installer file.

4. Type the command with the security mode command parameter set to SQL and the password set to a password of your choice.

   Note In this example, the password 1234 is used.

```
setup.exe SECURITYMODE=SQL SAPWD="1234"
```

5. After the installer has finished, restart your computer.

This completes your installation of MSDE using the setup.exe installer. In the next section, you verify that the Northwind database is installed.

Northwind Database Installation

Some of the tutorials in this documentation show you how to display a Crystal report whose data configuration uses a programmatic logon to the Northwind database on a secure SQL Server.

These tutorials include the following:

Logging onto a Secure SQL Server Database (CrystalReportViewer Object Model)

Logging onto a Secure SQL Server Database Using SQL Authentication (ReportDocument Object Model

Logging onto a Secure SQL Server Database Using Integrated Security (ReportDocument Object Model)

Reduced-Code Secure Database Logon in a Web Site

Connecting to IDataReader

In this section, you verify that you have installed the Northwind sample database that is provided with SQL Server or MSDE for use with these tutorials. Before you start, check the following:

If you use SQL Server, Northwind may already be installed. To verify this, check that Northwind is one of the databases installed on your system.

If you have just installed MSDE by following the instructions in the previous section, see the instructions in this section to install Northwind.

Visual Studio version	Path to instnwnd.sql installer script
Visual Studio .NET 2002	C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\Samples\Setup\instnwnd.sql
Visual Studio .NET 2003	Install a copy of the instnwnd.sql installer script to your file directory, by following the instructions in this section. See "To download the latest version of the instnwnd.sql installer script."
Visual Studio 2005	Install a copy of the instnwnd.sql installer script to your file directory, by following the instructions in this section. See "To download the latest version of the instnwnd.sql installer script."

To download the latest version of the instnwnd.sql installer script

1. Go to the Microsoft SQL download website: http://www.microsoft.com/sql/downloads/.
2. Locate the latest version of the Northwind database installer script (SQL2000SampleDb.msi).
3. Download and run the SQL2000SampleDb.msi installer. That creates a copy of the instnwnd.sql installer script on your local hard drive in the following file directory:

   ```
   C:\Program Files\Microsoft SQL Server 2000 Sample Database Scripts\
   ```

You now have a copy of the instnwnd.sql installer script. Proceed to the next step procedure for installation instructions.

To install the Northwind database in MSDE

1. Go to the command prompt.
2. Locate the Northwind installer script named instnwnd.sql (see table).
3. From the command prompt, change directory to the path of the installer script.

   Type the following command to install the database.

   ```
   osql -U sa -P [password] -i instnwnd.sql
   ```

   Note For [password], type the system administrator password that you created earlier in MSDE Installation with Windows or SQL Server Authentication.
4. Press ENTER.

   The Northwind database installs.

   Note The install process may take several minutes.

To verify the connection to the Northwind database

1. Launch Visual Studio 2005.
2. From the **View** menu, click **Server Explorer**.
3. In **Server Explorer**, right-click **Data Connections**, and then click **Add Connection...**

   You can now test either Windows Authentication or SQL Server Authentication.
4. If this is the first time you have added a connection in Visual Studio 2005 the **Change Data Source** window will appear. Select **Microsoft SQL Server** and click **Continue**.
   a) To test Windows Authentication, do the following:

      Type the name of your MSDE server (typically your computer name).

      Select the "Use Windows NT Integrated security" option.

      Click the **Select the database on the server** dropdown list, and then select "Northwind."

   b) To test SQL Server Authentication, do the following:

      Type the name of your MSDE server (typically your computer name).

      Select the "Use a specific name and password" option.

      Type the following user name: `sa`.

      Type the system administrator password that you created earlier in MSDE Installation with Windows or SQL Server Authentication.

      Click the **Select the database on the server** dropdown list, and then select "Northwind."
5. Click **Test Connection**, to verify that your installation of Northwind with either Windows Authentication or SQL Server Authentication is successful.

If you have chosen to use SQL Server Authentication, for security you should not use the system administrator account. Instead, to address security concerns, create a database account with more limited access for use by your web application. In the following section you learn how to create this limited access account for addressing the Northwind database.

Security: Creating a Limited Access Database Account

Note This information applies to database configurations that use only SQL Server Authentication. Windows Authentication does not require a limited access database account, because it uses Integrated Security.

When you access a database from a web application, you need to address security issues. In particular, it is important that the account used to access the database is limited in scope to only those functions that are strictly required by the web application.

For example, the Northwind database is used in the Crystal Reports documentation tutorials to display reports that are based on its Customers table and Orders table. Therefore, the limited access database account that is required to connect to the Northwind database would need only two permissions:

Permission to access the Customers and Orders table of the Northwind database.

Within these two tables, permission to SELECT records (but not INSERT, UPDATE, or DELETE).

In this section, you learn how to create limited access database account.

Note The creation of this limited access database account does not prevent you from accessing this database with the 'sa' system administrator account for full control. Instead, you create a secondary account for use only by your web application.

These instructions are equally applicable to both MSDE and SQL Server.

To create a limited access database account for the Northwind database

1. Type the following command to logon to your MSDE or SQL Server.

   Customize the following values in your command line arguments:

   Use the system administrator password where you see this placeholder: [password].

   Use the database server name where you see this placeholder: [serverName]

   ```
   osql -U sa -P [password] -S [serverName]
   ```

2. Type "USE master" to switch to the master database, then on the following line type "GO", and then press Enter.

   ```
   USE master
   GO
   ```

   Note "GO" alerts the server to process all instructions on the preceding line or lines.

3. Run the database script "sp_addlogin" and include as command line arguments the name that you want to use for your limited access database account, the password, and the database to which it applies.

   Customize the following values in your command line arguments:

   Create a limited access database account name, such as "limitedPermissionAccount".

Create a new password to be used by the limited access account, where you see this placeholder: [new_password]. Write down this password, as you will need it for some of the tutorials.

```
sp_addlogin 'limitedPermissionAccount','[new_password]','Northwind'
GO
```

4. Switch to the Northwind database.

```
USE Northwind
GO
```

5. Run the database script "sp_grantdbaccess" and pass in the name of the new limited access database account that you have created.

```
sp_grantdbaccess 'limitedPermissionAccount'
GO
```

6. For the Customers table, grant the SELECT privilege for the new limited access database account that you have created for two tables, Customers and Orders.

```
GRANT SELECT ON Customers
TO limitedPermissionAccount
GO
GRANT SELECT ON Orders
TO limitedPermissionAccount
GO
```

7. Type exit to exit the osql command line, and then press Enter.

```
exit
```

You have successfully created a limited access account and granted it SELECT access only, for the Customers and Orders table only, of the Northwind database.

To verify the connection to the Northwind database through the limited access account

1. Launch Visual Studio 2005.
2. From the **View** menu, click **Server Explorer**.
3. In **Server Explorer**, right-click **Data Connections**, and then click **Add Connection...**
4. If this is the first time you have added a connection in Visual Studio 2005 the **Change Data Source** window will appear. Select **Microsoft SQL Server** and click **Continue**. On the **Connection** tab, do the following:

   Type the name of your MSDE server (typically your computer name).

   Select the "Use SQL Server Authentication" option.

   Type the name of the limited access account that you created in the previous step procedure.

   Type the password for the limited access account that you created in the previous step procedure.

Click the **Select the database on the server** dropdown list, and then select "Northwind."

5. Click **Test Connection**, to verify that your installation of Northwind succeeded.

   This completes the setup of your limited access account to connect to the Customers table of the Northwind database.

Sample Reports' Directory

Some of the tutorials rely on sample reports that are installed with Crystal Reports for Visual Studio .NET.

If you have installed Crystal Reports with the default settings and file paths, sample reports are in the directories shown below.

Crystal Reports Version	Path to Sample Reports
Crystal Reports for Visual Studio .NET 2002	C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Reports\Feature Examples\ and C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Reports\General Business\
Crystal Reports 9	C:\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Reports\Feature Examples\ and C:\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Reports\General Business\
Crystal Reports for Visual Studio .NET 2003	C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Reports\Feature Examples\ and C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Reports\General Business\
Crystal Reports 10	C:\Program Files\Crystal Decisions\Crystal Reports 10\Samples\En\Reports\Feature Examples\ and C:\Program Files\Crystal Decisions\Crystal Reports 10\Samples\En\Reports\General Business
Crystal Reports 11	C:\Program Files\Business Objects\Crystal Reports 11\Samples\En\Reports\Feature Examples\ and C:\Program Files\Business Objects\Crystal Reports 11\Samples\En\Reports\General Business
Crystal Reports for Visual Studio 2005	C:\Program Files\Microsoft Visual Studio 8\Crystal Reports\Samples\En\Reports\Feature Examples\ and C:\Program Files\Microsoft Visual Studio 8\Crystal Reports\Samples\En\Reports\General Business\

Tutorials' Sample Code Directory

In this online help, the tutorials provide detailed step procedures that guide you through the completion of complex tasks. The tutorials are also available as completed sample code..

If you have installed Crystal Reports with the default settings and file paths, sample code is in the directories shown below.

Crystal Reports for Visual Studio 2005 includes a sample code installer. When you run the installer file, the sample code will be extracted to the directory of your choice, and the English version of the Xtreme sample database will be installed on your system.

Crystal Reports version	Path to tutorials' sample code
Crystal Reports for Visual Studio .NET 2002	Go to the support website for Crystal Reports and locate the "Download product documentation" link. Navigate to the page and download the tutorials' sample code for Crystal Reports .NET SDK.
Crystal Reports 9	Go to the support website for Crystal Reports and locate the "Download product documentation" link. Navigate to the page and download the tutorials' sample code for Crystal Reports .NET SDK.
Crystal Reports for Visual Studio .NET 2003	Go to the support website for Crystal Reports and locate the "Download product documentation" link. Navigate to the page and download the tutorials' sample code for Crystal Reports .NET SDK.
Crystal Reports 10	Go to the support website for Crystal Reports and locate the "Download product documentation" link. Navigate to the page and download the tutorials' sample code for Crystal Reports .NET SDK.
Crystal Reports 11	C:\Program Files\Business Objects\Crystal Reports 11\Developer Files\Help\En\CR_NET_SDK_Tutorial_Sample_Code.zip
Crystal Reports for Visual Studio 2005	C:\Program Files\Microsoft Visual Studio 8\Crystal Reports\Samples\en\Code\TutorialSampleCodeProjects.msi

Note In Visual Studio .NET 2002 or 2003, Web projects from external sources need to be imported.

Viewers' Virtual Directory

Crystal Reports relies on a virtual directory to access viewers for display. The virtual directory and its underlying file path are unique for each version of Crystal Reports; that way, succeeding versions of Crystal Reports on the same machine work without conflict.

To locate the viewers' virtual directory

1. In Control Panel, double-click **Administrative Tools**, and then double-click **Internet Services Manager**.
2. In the **Internet Information Services** dialog box, expand the top nodes, and then expand the **Default Web Site** node.
3. Locate the virtual directory folder as specified on the table below.
4. Right-click on the virtual directory folder to select **Properties**.
5. In the **Properties** dialog box, confirm that the **Local Path** is correctly configured for your version of Crystal Reports or Visual Studio.

If you have installed Crystal Reports with the default settings and file paths, the viewers' virtual directory is configured as shown below.

Version	Viewers' virtual directory name	File path
Crystal Reports for Visual Studio .NET 2002	CrystalReportWebFormViewer	C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Viewers
Crystal Reports 9	crystalreportviewers	C:\Program Files\Common Files\Crystal Decisions\2.0\crystalreportviewers
Crystal Reports for Visual Studio .NET 2003	CrystalReportWebFormViewer2	C:\Program Files\Microsoft Visual Studio .NET 2003\Crystal Reports\Viewers
Crystal Reports 10	crystalreportviewers10	C:\Program Files\Common Files\Crystal Decisions\2.5\crystalreportviewers10
Crystal Reports 11	crystalreportviewers11	C:\Program Files\Common Files\Business Objects\3.0\crystalreportviewers11
Crystal Reports for Visual Studio 2005	CrystalReportWebFormViewer3	**File path when using ASP.NET Development Server** [Windows folder]\Microsoft.NET\Framework\v2.0.51014\ASP.NETClientFi

		les\CrystalReportWebFormViewer3 **File path when using IIS** C:\Inetpub\wwwroot\aspnet_client\system_web\2_0_50526\CrystalReportWebFormViewer3

Location of Xtreme Sample Database

Some of the tutorials rely on the xtreme.mdb Microsoft Access database that is installed with Crystal Reports for Visual Studio.

The database is stored in your file directory, and is accessed through ODBC by an ODBC DSN configuration in the Data Sources (ODBC) control panel.

If you have installed Crystal Reports with the default settings and file paths, the xtreme.mdb database is located in the directory shown below.

Crystal Reports version	Path to xtreme.mdb
Crystal Reports for Visual Studio .NET 2002	C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Database\xtreme.mdb
Crystal Reports 9	C:\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases\xtreme.mdb
Crystal Reports for Visual Studio .NET 2003	C:\Program Files\Microsoft Visual Studio .NET 2003\Crystal Reports\Samples\Database\xtreme.mdb
Crystal Reports 10	C:\Program Files\Crystal Decisions\Crystal Reports 10\Samples\En\Databases\xtreme.mdb
Crystal Reports 11	C:\Program Files\Business Objects\Crystal Reports 11\Samples\En\Databases\xtreme.mdb
Crystal Reports for Visual Studio 2005	C:\Program Files\Microsoft Visual Studio 8\Crystal Reports\Samples\En\Databases\xtreme.mdb

ODBC DSN Entry for Xtreme Sample Database

A number of tutorials, which are found both in this documentation and in other help materials on the Business Objects Website, rely on the xtreme.mdb Microsoft Access database that is installed with Crystal Reports for Visual Studio.

The database is stored in your file directory, and it is accessed through ODBC by an ODBC DSN configuration in the Data Sources (ODBC) control panel.

In the previous section, you verified the location of the database. In this section, you verify the DSN configuration.

To verify the ODBC DSN configuration for connecting to the xtreme.mdb database

1. In Control Panel, select **Administrative Tools**.
2. Select **Data Sources (ODBC)**.
3. Select the tab **System DSN**.
4. Locate the corresponding System DSN entry for your version of Crystal Reports from the table below.
5. Confirm that the DSN entry exists on your system, and that it contains the correct name and path to the database.
6. If the DNS entry does not exist, create a new System DSN entry according to the table below.

Crystal Reports version	System DSN name	Path to xtreme.mdb
Crystal Reports for Visual Studio .NET 2002	Xtreme Sample Database	C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Database\xtreme.mdb
Crystal Reports 9	Xtreme Sample Database 9	C:\Program Files\Crystal Decisions\Crystal Reports 9\Samples\En\Databases\xtreme.mdb
Crystal Reports for Visual Studio .NET 2003	Xtreme Sample Database 2003	C:\Program Files\Microsoft Visual Studio .NET\Crystal Reports\Samples\Database\xtreme.mdb
Crystal Reports 10	Xtreme Sample Database 10	C:\Program Files\Crystal Decisions\Crystal Reports 10\Samples\En\Databases\xtreme.mdb
Crystal Reports 11	Xtreme Sample Database 11	C:\Program Files\Business Objects\Crystal Reports 11\Samples\En\Databases\xtreme.mdb

Crystal Reports for Visual Studio 2005	Xtreme Sample Database 2005	C:\Program Files\Microsoft Visual Studio 8\Crystal Reports\Samples\En\Database\xtreme.mdb

Project Setup

This section is a key learning point for both advanced and intermediate developers. It demonstrates the recommended best practices to follow to create and configure a new Windows project or Web project/site with Crystal Reports for Visual Studio .NET 2002 or 2003, and Crystal Reports for Visual Studio 2005.

The new project that you create also serves as a prerequisite for the tutorials that are provided with this online help.

Project Setup in Visual Studio 2005

In Crystal Reports for Visual Studio 2005, the structure of Web and Windows applications is no longer parallel:

Web applications are no longer built as projects. They are now built as Web Sites.

This means that for Web applications the project metaphor is gone. The contents of a Web Site folder are now simpler than that of a Windows project. In particular, the configuration information previously spread across project and global files has either been removed altogether or else relocated to the Web.config file.

Windows applications continue to be built as projects.

Web Site Setup in Visual Studio 2005

This section is a key learning point for both advanced and intermediate developers. It demonstrates the recommended best practices to follow when you create and configure a new Web Site with Crystal Reports for Visual Studio 2005.

This section demonstrates Web Site Setup using a coding model. As part of this setup, you will go into the code-behind class and enter code. This code-based Web Site that you create serves as a prerequisite for the coding tutorials that are provided with this online help.

The procedures in this section must be completed in succession:

Creating a New Web Site

Preparing the Web Form

Adding a CrystalReportViewer Control

Note If you do not plan to use the coding model, but want to learn instead how to build a Web Site using the reduced-code, tag-based development model provided with Visual Studio 2005, go to the tutorial Reduced-Code Web Site Setup with Crystal Reports Using Smart Tasks.

Creating a New Web Site in Visual Studio 2005

Before you create a Web Site, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a Web Site in Crystal Reports for Visual Studio 2005

1. Launch Visual Studio 2005.
2. From the **File** menu, click **New Web Site**.
3. In the **New Web Site** dialog box, click **ASP.NET Web Site**.

4. In the **Location** dropdown, select **File System**.
5. In the **Language** dropdown, select the coding language that you wish to use.
6. In the **Location** text field enter the directory path "C:\WebSites\", followed by the name of your project.

   ```
   C:\WebSites\MyProjectName
   ```

   Note In Project Setup for Visual Studio .NET 2002 or 2003 a Pascal naming convention was recommended (setting the first letter of the project name to uppercase). In Visual Studio 2005, because the namespace is no longer related to the Web Site name, you may use any case that you prefer.
7. Click **OK**.

Preparing the Web Form in Visual Studio 2005

In this section you configure the code-behind class for the Web form.

Note The terms "Web form", "ASPX page" and "Default.aspx" are used interchangeably.

To prepare the Web Form in Crystal Reports for Visual Studio 2005

1. From the Solution Explorer, double click on Default.aspx to open the Web form.

   Note In Visual Studio .NET 2002 or 2003, the .aspx file would open in Design View by default. In Visual Studio 2005, an .aspx file will expose the markup by default instead of the designer.
2. From the **View** menu, click **Code**.

   The code-behind class opens. The class is named _Default class. The class file is named Default.aspx.cs or Default.aspx.vb.

   Note If your Default.aspx page was created with inline code, it did not place its code into a separate file. In that case, delete the ASPX page and recreate it. When creating the ASPX page, select the "Place code in separate file" checkbox.
3. If you are writing this class in Visual Basic, type "Option Strict On" at the top of the class.

   Note As a best practice, it is recommended that you set Option Strict On at the start of every Visual Basic class in your Web Site. When you write code, it forces the use of best practices with strongly typed variable declarations and valid casting, both of which are checked at compile time. Compile-time checks that are strictly enforced can reduce run-time exceptions.

Next, you add a private helper method that is used as the designated location for all code that configures Crystal Reports for the class.

To add a private helper method for Crystal Reports configuration code

1. Within the class, add a new private scope helper method, with no return value, named `ConfigureCrystalReports()`.

[Visual Basic]

```
Private Sub ConfigureCrystalReports()


End Sub
```

[end]

[C#]

```
private void ConfigureCrystalReports()

{

}
```

[end]

The `ConfigureCrystalReports()` method enables users to interact with the report at runtime. It also controls programmatic interaction with the report.

Next, you add a Page_Init event handler from which to call the `ConfigureCrystalReports()` method. Calling the `ConfigureCrystalReports()` method from this event handler guarantees that the Crystal report configuration code runs during the page initialization event.

To add a Page_Init event handler to the code-behind class

Typically the Page_Load event handler is used to enter Web Form configuration code in an ASP.NET Web application so that the code will be called during the Page.Load event. However, the Crystal report configuration code needs to be called earlier, during the Page.Init event. If you are coding in Visual Basic, do the following:

At the top left drop-down list of the Code view, select Page Events.

Note This causes the top right drop-down list to be populated with all available events for this control.

From the top right drop-down list, choose the **Init** event.

The following event handler is added to your code-behind class:

[Visual Basic]

```
Private Sub Page_Init(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Init


End Sub
```

[end]

1. If you are coding in C#, enter the Page_Init event handler using the exact syntax shown.

[C#]

```
private void Page_Init(object sender, EventArgs e)

{


}
```

[end]

Note In a C# Web form in Visual Studio 2005, any Page_Init, Page_Load or Page_PreRender event handler in the code-behind class is wired automatically to the Init, Load or PreRender event. The event handler signature must match exactly in order to be called.

This feature occurs when the Page directive, found at the top of the ASPX page in HTML view, has its AutoEventWireup parameter set to true.

Previous versions of Visual Studio .NET and Visual Basic Web forms in Visual Studio 2005 always set the AutoEventWireup Page directive to False, but in C# Web forms in Visual Studio 2005 the AutoEventWireup Page directive is set to True by default.

2. Finally, within the Page_Init event handler for either Visual Basic or C#, enter a call to the `ConfigureCrystalReports()` helper method.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

3. From the **File** menu, click **Save All**.

Adding a CrystalReportViewer Control in Visual Studio 2005

You are now ready to add the CrystalReportViewer control.

To add a CrystalReportViewer control

1. Open the Default.aspx page.
2. Click the **Design** button at the bottom of the form view.
3. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
4. Drag and drop the **CrystalReportViewer** control onto the Web Form.

   The CrystalReportViewer control displays a new GUI feature known as a Smart Task panel on the upper-right corner of the control.

   Note This is part of the reduced-code development model provided with ASP.NET version 2.0. The Smart Task panel for the CrystalReportViewer control simplifies configuration of functionality such as report binding and control layout using GUI settings. Any selections that you make in the Smart Task panel are then auto-generated as tag-based settings within the ASPX page. Since you are currently working with the code-based development model, the Smart Task panel is not used during this setup.
5. If the Smart Task panel "CrystalReportViewer Tasks" is open, press **Esc** on your keyboard to close it.
6. Click the **CrystalReportViewer** control to select it.
7. From the **Properties** window, set the **ID** property:

   For Visual Basic Web Sites, set the **ID** property to myCrystalReportViewer.

   For C# Web Sites, set the **ID** property to crystalReportViewer.
8. From the **File** menu, click **Save All**.

In the next section you add namespace references for Crystal Reports namespaces.

To add Imports/Using statements to reference namespaces

1. Open the Default.aspx page.
2. From the **View** menu, click **Code**.

   The code-behind class for the Web Form appears.
3. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class containing the following Crystal Reports namespaces.

[Visual Basic]

```
Imports CrystalDecisions.CrystalReports.Engine
Imports CrystalDecisions.Shared
```

[end]

[C#]

```
using CrystalDecisions.CrystalReports.Engine;
using CrystalDecisions.Shared;
```

[end]

> **Note** The classes of these two assemblies are commonly used in all tutorials. For any additional assemblies that you may occasionally require in specific tutorials, you will be directed to add them during that tutorial.

You have now completed Project Setup. However, a number of tutorials have additional setup requirements, such as adding a sample report.

Windows Project Setup in Visual Studio 2005

This section is a key learning point for both advanced and intermediate developers. It demonstrates the recommended best practices to follow to create and configure a new Windows project with Crystal Reports for Visual Studio 2005.

This section demonstrates Windows project setup using a coding model. As part of this setup, you will go into the Form1 class and enter code. This code-based Windows project that you create serves as a prerequisite for the coding tutorials that are provided with this online help.

If you wish to build a Windows project using the reduced-code model that is provided with Visual Studio 2005, go to the tutorial Reduced-Code Windows Project Setup with Crystal Reports Using Smart Tasks.

The procedures in this section must be completed in succession:

Creating a New Windows Project

Applying Standard Visual Basic Project Settings

Preparing the Windows Form

Adding a CrystalReportViewer Control

Creating a New Windows Project in Visual Studio 2005

Before you create a Windows project, verify that Crystal Reports for Visual Studio 2005 has been installed on your system.

To set up a Windows project in Crystal Reports for Visual Studio 2005

1. Launch Visual Studio 2005.
2. From the **File** menu, select **New**, and then click **Project**.
3. In the **New Project** dialog box, select a language folder for C# or Visual Basic from the **Project Types** list.
4. From the **Templates** list, click **Windows Application**.
5. In the **Name** field, replace the default project name with the name of your project.

   Use a Pascal naming convention where you set the first letter of the project name to uppercase, because the project name also becomes the namespace name for the assembly generated from the project.
6. Click **OK**.

Applying Standard Visual Basic Project Settings

You must make a minor modification to the project settings in a Visual Basic project, to configure the project to work with the tutorials in this documentation.

If your project is developed in C#, continue to Preparing the Windows Form.

To modify project settings for a Visual Basic project in Crystal Reports for Visual Studio 2005

1. In **Solution Explorer**, right-click the bold project name that is below the solution name, and then select **Properties**.
2. In the **Properties** view, click the **Compile** tab.
3. On the **OptionStrict** list, click **On**.
4. Close the **Properties** view.
5. From the **File** menu, click **Save All**.

   Note As a best practice, it is recommended that you enable OptionStrict at the start of any Visual Basic project. When you write code, it forces the use of best practices with strongly typed variable declarations and valid casting, both of which are checked at compile time. Compile-time checks that are strictly enforced can reduce run-time exceptions.

Preparing the Windows Form in Visual Studio 2005

Traditional Visual Basic 6 Windows applications typically defined a default form under the name Form1. In keeping with that pattern in a Windows project, you use the same default form name, Form1 with a .cs or .vb extension that depends on the language you use.

To prepare the Windows Form in Crystal Reports for Visual Studio 2005

1. If **Form1** is not already displayed in the main window, double-click **Form1** in **Solution Explorer**.

   Form1 opens in the Designer.
2. From the **View** menu, click **Code**.

   The code view of the Form1 class appears. The display of this class depends on whether your Windows application is coded in Visual Basic or C#.

   In C#, the Form1 class displays the following:

The class signature.

A constructor (Form1).

In Visual Basic, the Form1 class displays the following:

The class signature (a Form1 class).

Note Additional methods and variables of the Form1 class are contained in a separate "partial" class. (This is true for both C# and Visual Basic.) Partial classes are explained later in Windows Project Setup.

Next, you add a private helper method that is used as the designated location for all the code that configures Crystal Reports for the class.

To add a private helper method for Crystal Reports configuration code

1. Add to this Form1 class a new private scope helper method with no return value, named `ConfigureCrystalReports()`.

[Visual Basic]

```
Private Sub ConfigureCrystalReports()


End Sub
```

[end]

[C#]

```
private void ConfigureCrystalReports()
{
}
```

[end]

Next you add a Form_Load event handler, and then put a call to `ConfigureCrystalReports()` within the Form_Load event handler. This will cause the `ConfigureCrystalReports()` method to run automatically when the form loads.

2. From the **View** menu, click **Designer**.
3. Double-click on Form1.

   You are returned to Code view. Because you double-clicked on Form1, a Form1_Load event handler is automatically generated in the Form1 class.
4. Within the Form1_Load event handler, enter a call to the `ConfigureCrystalReports()` method.

[Visual Basic]

```
ConfigureCrystalReports()
```

[end]

[C#]

```
ConfigureCrystalReports();
```

[end]

5. From the **File** menu, click **Save All**.

Adding a CrystalReportViewer Control in Visual Studio 2005

To display reports on a Windows Form, you add a CrystalReportViewer control.

To add a CrystalReportViewer control in Crystal Reports for Visual Studio 2005

1. Open Form1 in Design view.
2. From the **Toolbox**, open the **Crystal Reports** node to locate the **CrystalReportViewer** control.
3. Drag and drop the **CrystalReportViewer** control onto the form.

   The CrystalReportViewer control displays a new GUI feature known as a Smart Task panel on the upper-right corner of the control.

   Note This is part of the reduced-code development model provided with .NET version 2.0. The Smart Task panel for the CrystalReportViewer control simplifies configuration of functionality such as report binding and control layout using GUI settings. Since you are currently working with the code-based development model, the Smart Task panel is not used during this setup.
4. If the Smart Task panel is open, click the **arrow toggle** on the upper-right corner of the **CrystalReportViewer** control to close the Smart Task panel.
5. Click the **CrystalReportViewer** control to select it.
6. From the **Properties** window, set the **Name** property:

   For Visual Basic Windows project, set the **Name** property to myCrystalReportViewer.

   For C# Windows project, set the **Name** property to crystalReportViewer.
7. From the **File** menu, click **Save All**.
8. From the **Build** menu, click **Build Solution**.

To confirm that ConfigureCrystalReports() has been added correctly

h) From the **View** menu, click **Other Windows**, and then click **Object Browser**.

9. In the **Object Browser**, expand the **[project name]** node, then the **[project name]** namespace node, and then click on the **Form1** class.

   The members of the Form1 class are displayed, which include both the CrystalReportViewer variable (from the Form1.Designer.cs partial class) and the `ConfigureCrystalReports()` method (from the Form1.cs partial class).

In Visual Studio 2005, Form1.cs (or .vb) and its corresponding file Form1.designer.cs (or .vb) compile together into a single Form1 class that shares their members.

This is possible because of partial classes, a new feature in .NET Framework version 2.0. A class modifier named "partial" can be added to a class signature. This modifier indicates that the class file is an addendum to an already existing class of the same name.

This has the following impact on Windows Forms:

In Visual Studio .NET 2002 or 2003, the Form1.cs (or .vb) file had many lines of auto-generated code, including class-level declarations for controls added to the Windows Form, as well as event and other configuration code for the Form and its controls. In Visual Studio 2005 this auto-generated code has been moved to the less visible Form1.designer.cs (or .vb) file, which is the Form1 partial class. This allows the original

Form1 class to access all of the auto-generated code from the Form1 partial class, while keeping itself free for additional code to be written by the developer.

To add Imports/Using Statements to reference namespaces

1. If the **References** folder is not visible in **Solution Explorer**, on the **Solution Explorer** toolbar, click **Show All Files** to display all project files.
2. In **Solution Explorer**, expand the **References** folder.
3. Verify that the following Crystal Reports assemblies have been added:

   CrystalDecisions.CrystalReports.Engine

   CrystalDecisions.Shared
4. Select Form1, and then from the **View** menu, click **Code**.

   The Code view of the Form1 class appears.
5. Above the class signature, add an `"Imports" [Visual Basic]` or `"using" [C#]` declaration to the top of the class containing the following Crystal Reports namespaces.

[Visual Basic]

```
Imports CrystalDecisions.CrystalReports.Engine
Imports CrystalDecisions.Shared
```

[end]

[C#]

```
using CrystalDecisions.CrystalReports.Engine;
using CrystalDecisions.Shared;
```

[end]

> **Note** To access various classes for Crystal Reports without a namespace prefix, you must declare those namespaces at the top of the class.

You have now completed Project Setup. However, a number of tutorials have additional setup requirements, such as adding a sample report.

Additional Setup Requirements

Now your Crystal Reports for Visual Studio 2005 project is set up, ready for you to code. However, you might have more setup work to do, depending on the tutorial or report binding scenario that you follow.

The table below shows which additional options to complete for each tutorial and report binding scenario.

If you are working on	Then in addition to Project Setup, complete the following:
Basic Web or Windows project setup with a sample report in Visual Studio .NET 2002 or 2003.	Add a Sample Report as an Embedded Report
Basic Web Site setup with a sample report in Visual Studio 2005.	Add a Sample Report as a Non-embedded Report in a Visual Studio 2005 Web Site
Basic Windows project setup with a sample report in Visual Studio 2005.	Add a Sample Report as an Embedded Report
Tutorial: Persisting the ReportDocument Object Model Using Session	If you are creating this tutorial as a Web Site in Visual Studio 2005, your report must be non-embedded. Go to: Add a Sample Report as a Non-embedded Report in a Visual Studio 2005 Web Site Otherwise, your report is embedded. Go to: Add a Sample Report as an Embedded Report
Tutorial: Exporting to Multiple Formats	If you are creating this tutorial as a Web Site in Visual Studio 2005, your report must be non-embedded. Go to: Add a Sample Report as a Non-embedded Report in a Visual Studio 2005 Web Site Otherwise, your report is embedded. Go to: Add a Sample Report as an Embedded Report After you have added a sample report, you must also do the following: Add a Class for Error Messages
Tutorial: Printing and Setting Print Options	If you are creating this tutorial as a Web Site in Visual Studio 2005, your report must be non-embedded. Go to: Add a Sample Report as a Non-embedded Report in a Visual Studio 2005 Web Site Otherwise, your report is embedded. Go to: Add a Sample Report as an Embedded Report After you have added a sample report, you

	must also do the following: Add a Class for Error Messages
Tutorial: Configuring Multilingual Client Support	If you are creating this tutorial as a Web Site in Visual Studio 2005, your report must be non-embedded. Go to: Add a Sample Report as a Non-embedded Report in a Visual Studio 2005 Web Site Otherwise, your report is embedded. Go to: Add a Sample Report as an Embedded Report
Report binding scenario: Binding to ReportSource (BusinessObjects Enterprise 11)	Adding Assembly References for the BOE SDK
Report binding scenario: Binding to an Embedded Report Class	Add a Sample Report as an Embedded Report
Report binding scenario: Binding to an Embedded Report Class Upcast to ReportDocument	Add a Sample Report as an Embedded Report
Report binding scenario: Binding to a Cached Embedded Report Class	Add a Sample Report as an Embedded Report
Report binding scenario: Binding to an Unmanaged RAS Server Using ReportDocument.Load() Method (RAS 10)	Adding Assembly References for the RAS SDK
Report binding scenario: Binding to an Unmanaged RAS Server Using ReportDocument.FileName Property (RAS 10)	Adding Assembly References for the RAS SDK
Report binding scenario: Binding to an Unmanaged RAS Server Using ReportClientDocument.Open() Method (RAS 9 and up)	Adding Assembly References for the RAS SDK
Report binding scenario: Binding to a Managed RAS Server Using ReportDocument.Load() Method	Adding Assembly References for the BOE SDK
Report binding scenario: Binding to a Managed RAS Server Using ReportDocument.FileName Property	Adding Assembly References for the BOE SDK
Report binding scenario: Binding to a Managed RAS Server Using ReportAppFactory.OpenDocument() Method	Adding Assembly References for the BOE SDK

Report binding scenario: Binding to InfoObject Cast as Report	Adding Assembly References for the BOE SDK

If none of the options apply, you can return to tutorials or report binding scenarios.

Add a Sample Report as an Embedded Report

In this section, you add a sample Crystal Reports .rpt file as an embedded report to your project.

Note This procedure is required by only some tutorials and report binding scenarios. This procedure is done only after the completion of Project Setup. To verify whether you need to complete this procedure, consult the table in Additional Setup Requirements.

Adding an Existing Report from the Sample Reports Directory

To add an existing report from the sample reports directory into a Windows or Web project

Note This procedure works only with a project that has been created from Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. From the **Project** menu, click **Add Existing Item**.
2. In the **Add Existing Item** dialog box, set **Files of type** to All Files (*.*).
3. Navigate to the **Hierarchical Grouping.rpt** file in the Feature Examples folder of the Crystal Reports sample reports directory.

   Note The Hierarchical Grouping report retrieves its data from the Access database xtreme.mdb.
4. Click the **Hierarchical Grouping.rpt** file to select it, and then click **Open**.

   The Hierarchical Grouping.rpt file is added to your project.

   The report that you have added to your project is embedded. This means that any report added to the project is embedded in the project assembly at compile time; you do not have to manually link reports to the project.

   For coding purposes, a report class is automatically generated that wraps (represents) the report file. In the next procedure, you instantiate this report class and bind it to the CrystalReportViewer control.

Binding the Embedded Report to the CrystalReportViewer Control

This section shows the simplest way to bind an embedded report, to enable your project to locate and display it.

To bind the embedded report to the CrystalReportViewer control

1. Open the Web or Windows Form in Design view.
2. From the **View** menu, click **Code** to view the code-behind class for this Web or Windows Form.
3. At the top of the class, add a new class-level declaration for the Hierarchical_Grouping report wrapper class, with the variable name hierarchicalGroupingReport. Set its access modifier to private.

[Visual Basic]

```
Private hierarchicalGroupingReport As Hierarchical_Grouping
```

[end]

[C#]

```
private Hierarchical_Grouping hierarchicalGroupingReport;
```

[end]

4. Within the `ConfigureCrystalReports()` method (which you added during one of the procedures in Project Setup), instantiate the report wrapper class.

[Visual Basic]

```
hierarchicalGroupingReport = New Hierarchical_Grouping()
```

[end]

[C#]

```
hierarchicalGroupingReport = new Hierarchical_Grouping();
```

[end]

5. On the next line, beneath the report instantiation, bind the ReportSource property of the CrystalReportViewer to the instantiated report class (variable name: hierarchicalGroupingReport).

[Visual Basic]

```
myCrystalReportViewer.ReportSource = hierarchicalGroupingReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = hierarchicalGroupingReport;
```

[end]

You are now ready to build and run the project.

To build and run the project

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, fix them now.
3. From the **Debug** menu, click **Start**.

   If no build errors appear, the project loads into your Web browser (Web) or Windows application (Windows) and displays the Web or Windows Form, with the Hierarchical Grouping Report generated on the form.
4. Return to Visual Studio, and then click **Stop** to exit from debug mode.

Add a Sample Report as a Non-embedded Report in a Visual Studio 2005 Web Site

In this section, you reference a sample Crystal Reports .rpt file as a non-embedded report.

Note This procedure is required by only some tutorials and report binding scenarios. This procedure is done only after the completion of Project Setup. To verify whether you need to complete this procedure, consult the table in Additional Setup Requirements.

Binding the Non-embedded Report to the CrystalReportViewer Control in a Visual Studio 2005 Web Site

This section shows the simplest way to bind a non-embedded report, to enable your project to locate and display it.

To bind the non-embedded report to the CrystalReportViewer control in a Visual Studio 2005 Web Site

Note This procedure works only with a project that has been created from Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. To locate the report, go to the **Hierarchical Grouping.rpt** file in the Feature Examples folder of the Crystal Reports sample reports directory.

   Note The Hierarchical Grouping report retrieves its data from the Access database xtreme.mdb.
2. Copy the file directory path of the report.
3. Open the Default.aspx Web form that contains the CrystalReportViewer control that you added in Project Setup in Visual Studio 2005.
4. From the **View** menu, click **Code**, to view the code-behind partial class for the Web form.
5. At the top of the class, add a new class-level declaration for the ReportDocument class, with the variable name hierarchicalGroupingReport. Set its access modifier to private.

[Visual Basic]

```
Private hierarchicalGroupingReport As ReportDocument
```

[end]

[C#]

```
private ReportDocument hierarchicalGroupingReport;
```

[end]

Note The ReportDocument class is a member of the CrystalDecisions.CrystalReports.Engine namespace. You added an `"Imports"` `[Visual Basic]` or `"using"` `[C#]` declaration for this namespace in Project Setup.

6. Within the `ConfigureCrystalReports()` method (which you added during one of the procedures in Project Setup), instantiate the ReportDocument class.

[Visual Basic]

```
hierarchicalGroupingReport = New ReportDocument()
```

[end]

[C#]

```
hierarchicalGroupingReport = new ReportDocument();
```

[end]

7. In the next line, call the `Load()` method of the ReportDocument instance and paste into it the file directory path that you copied.

   Note The sample below shows the file directory path from Visual Studio 2005.

[Visual Basic]

```
hierarchicalGroupingReport.Load("C:\Program Files\Microsoft Visual
Studio 8\Crystal Reports\Samples\En\Reports\Feature
Examples\Hierarchical Grouping.rpt")
```

[end]

[C#]

```
hierarchicalGroupingReport.Load(@"C:\Program Files\Microsoft Visual
Studio 8\Crystal Reports\Samples\En\Reports\Feature
Examples\Hierarchical Grouping.rpt");
```

[end]

8. On the next line, beneath the report loading, bind the ReportSource property of the CrystalReportViewer to the ReportDocument instance.

[Visual Basic]

```
myCrystalReportViewer.ReportSource = hierarchicalGroupingReport
```

[end]

[C#]

```
crystalReportViewer.ReportSource = hierarchicalGroupingReport;
```

[end]

You are now ready to build and run the project.

To build and run the project

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, fix them now.
3. From the **Debug** menu, click **Start**.

   Note If this is the first time you have started debugging in Visual Studio 2005, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.

4. Click **OK** to enable debugging.

   If no build errors appear, the project loads into your Web browser and displays the Web page, with the Hierarchical Grouping Report generated on the page.

5. Return to Visual Studio 2005, and then click **Stop** to exit from debug mode.

Add a Class for Error Messages

In this section, you create a class named MessageConstants containing five string constants:

SUCCESS

FAILURE

NOT_ALLOWED

FORMAT_NOT_SUPPORTED

NO_MATCHES_FOUND

> **Note** This procedure is required by only some tutorials and report binding scenarios. This procedure is done only after the completion of Project Setup. To verify whether you need to complete this procedure, consult the table in Additional Setup Requirements.

Why build a MessageConstants Class?

In .NET projects, you use try/catch blocks to trap successes and failures when you call the underlying business logic. After the method call is made, a success or failure message is assigned (in general, to a Label control) to display the result. The success message is assigned in the try block, and the failure message is assigned in the catch block. These success and failure messages should be created as string constants, in a class that is devoted to messages.

Other constants that you create in this procedure are used when an attempted activity is not allowed, a format cannot be found, and no matches are found for a submitted value.

> **Note** If the general information that is provided by the constant is not enough, you can append an additional message (for example, the Message property of Exception) after the constant.

Creating the MessageConstants class

In this section, you create the MessageConstants class.

To create the MessageConstants class

> **Note** This procedure works only with a project that has been created from Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. In a Windows project, from the **Project** menu, click **Add Class**.
2. In a Web Site, from the **WebSite** menu, click **Add New Item**.
3. In **Add New Item** dialog box, in the **Name** field enter MessageConstants.

   > **Note** The appropriate language-specific suffix, either .vb or .cs, is added automatically when you create the file.
4. Click **Add**.

   In Visual Studio 2005, you may be asked to place this class in a Code directory. If so, click Yes.

[Visual Basic]

```
    Public Class MessageConstants

    End Class
```

[end]

[C#]

```
    using System;


    namespace MyProjectName

    {

        public class MessageConstants

        {

            public MessageConstants()

            {

            }

        }

    }
```

[end]

5. Delete the default constructor method that is provided in the C# version of the code.
6. Create a public string constant named SUCCESS and assign to it the message: "The action was successful."

[Visual Basic]

```
    Public Const SUCCESS As String = "The action was successful."
```

[end]

[C#]

```
    public const string SUCCESS = "The action was successful.";
```

[end]

7. Create a public string constant named FAILURE and assign to it the message: "The action was not successful: "

[Visual Basic]

```
    Public Const FAILURE As String = "The action was not successful: "
```

[end]

[C#]

```
    public const string FAILURE = "The action was not successful: ";
```

[end]

   This string ends with a colon and a space so that the exception message can be appended to the constant, when it is assigned in a catch block.

8. Create a public string constant named NOT_ALLOWED and assign to it the message: "You are not allowed to do this action."

[Visual Basic]

```
    Public Const NOT_ALLOWED As String = "You are not allowed to do this action."
```

[end]

[C#]

```
public const string NOT_ALLOWED = "You are not allowed to do this action.";
```

[end]

9. Create a public string constant named FORMAT_NOT_SUPPORTED and assign to it the message: "That format is not supported."

[Visual Basic]

```
Public Const FORMAT_NOT_SUPPORTED As String = "That format is not
supported."
```

[end]

[C#]

```
public const string FORMAT_NOT_SUPPORTED = "That format is not supported.";
```

[end]

10. Create a public string constant named NO_MATCHES_FOUND and assign to it the message: "No matches were found for the value submitted."

[Visual Basic]

```
Public Const NO_MATCHES_FOUND As String = "No matches were found for the
value submitted."
```

[end]

[C#]

```
public const string NO_MATCHES_FOUND = "No matches were found for the value
submitted.";
```

[end]

You are now ready to build and run the project.

To build and run the project

1. From the **Build** menu, click **Build Solution**.
2. If you have any build errors, fix them now.
3. From the **Debug** menu, click **Start**.

   Note If this is the first time you have started debugging in Visual Studio 2005, a dialog box appears and states that the Web.config file must be modified. Click the OK button to enable debugging.
4. Click **OK** to enable debugging.

   If no build errors appear, the project loads into your Web browser (Web) or Windows application (Windows) and displays the Web or Windows Form with the Hierarchical Grouping Report generated on the form.
5. Return to Visual Studio 2005, and then click **Stop** to exit from debug mode.

You have created the MessageConstants class. These constants can now be called from anywhere within the project.

Adding Assembly References for the Report Application Server (RAS) SDK

This procedure may be an additional setup requirement to the procedures in Project Setup, depending on which tutorial or binding scenario you follow. To verify whether you must meet this requirement, consult the table in Additional Setup Requirements.

At the beginning of a .NET project, it is common practice to reference assemblies that contain the class libraries to code against.

The following procedure shows you how to add the necessary assemblies, to work with the Report Application Server (RAS) SDK.

To add assemblies for the RAS SDK

Note This procedure works only with a project that has been created from Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. In **Solution Explorer**, right-click the **References** folder and select **Add Reference**.

   In the Add Reference dialog box that appears, you can see three tabs: .NET, COM, and Projects. The .NET tab adds .NET assemblies, the COM tab adds COM objects, and the Projects tab adds other uncompiled Visual Studio 2005 projects, which are located on your hard drive.

   You use only the .NET tab. It displays all .NET assemblies that have been installed on your system.

2. If you have the RAS SDK installed, select the following CrystalDecisions.ReportAppServer assemblies:

   CrystalDecisions.ReportAppServer.ClientDoc

   CrystalDecisions.Controllers

   Note It is possible to have multiple versions of Crystal products on your machine. In that case, you see duplicate sets of assemblies in this window, but the version numbers would differ. Be careful to only add assemblies that share the same version number. You can choose which version you prefer to work with.

3. On the right side, click **Select**.

   This moves the selected assemblies into the selected components view at the bottom of the Add Reference dialog box.

4. Click **OK** to add these assemblies as references to your project.

   Notice that the assemblies are added into the References folder in Solution Explorer.

5. Collapse the References folder so that the remaining items in this project are visible in **Solution Explorer**.

   Note If you are using the Crystal Reports Server or BusinessObjects Enterprise SDK in your project, go to Adding Assembly References for the Crystal Reports Server or BusinessObjects Enterprise SDK.

Adding Assembly References for the Crystal Reports Server or BusinessObjects Enterprise SDK

This procedure may be an additional setup requirement to the procedures in Project Setup, depending on which tutorial or binding scenario you follow. To verify whether you must meet this requirement, consult the table in Additional Setup Requirements.

At the beginning of a .NET project, it is common practice to reference assemblies that contain the class libraries to code against.

The following procedure shows you how to add the necessary assemblies to work with the Crystal Reports Server or BusinessObjects Enterprise SDK.

To add assemblies for the Crystal Reports Server or BusinessObjects Enterprise SDK

Note This procedure works only with a project that has been created from Project Setup. Project Setup contains specific namespace references and code configuration that is required for this procedure, and you will be unable to complete the procedure without that configuration. Therefore, before you begin this procedure, you must first follow the steps in Project Setup.

1. In **Solution Explorer**, right-click the **References** folder and click **Add Reference**. This brings up the Add Reference dialog box.

   In the Add Reference dialog box that appears, you can see three tabs: .NET, COM, and Projects. The .NET tab adds .NET assemblies, the COM tab adds COM objects, and the Projects tab adds other uncompiled Visual Studio 2005 projects, which are located on your hard drive.

   You use only the .NET tab. It displays all .NET assemblies that have been installed on your system.

2. If you have the Crystal Reports Server or BusinessObjects Enterprise SDK installed, then the RAS SDK is also included. Select the following CrystalDecisions.ReportAppServer assemblies:

   CrystalDecisions.ReportAppServer.ClientDoc

   CrystalDecisions.ReportAppServer.Controllers

   The managed RAS server is included as a server within Crystal Reports Server or BusinessObjects Enterprise.

3. Select also the following CrystalDecisions.Enterprise assemblies:

   CrystalDecisions.Enterprise.Framework

   CrystalDecisions.Enterprise.InfoStore

   CrystalDecisions.Enterprise.Desktop.Report

   Note It is possible to have multiple versions of Crystal products on your machine. In that case, you see duplicate sets of assemblies in this window, but the version numbers would differ. Be careful to only add assemblies that share the same version number. You can choose which version you prefer to work with.

4. On the right side, click **Select**.

This moves the selected assemblies into the selected components view at the bottom of the Add Reference dialog box.

5. Click **OK** to add these assemblies as references to your project.

   Notice that the assemblies are added into the References folder in Solution Explorer.

6. Collapse the References folder so that the remaining items in this project are visible in **Solution Explorer**.

Multilingual Client Support

Crystal Reports for Visual Studio 2005 includes support for multilingual Web and Windows clients, through dynamic localization.

Dynamic localization allows users to view ToolTips and other content of the CrystalReportViewer control in their preferred language.

Language Resources

On the Web server or Windows client, you can access language resources for the CrystalReportViewer control from two possible sources:

Default language resource DLLs, which include the following:

English (en)

French (fr)

German (de)

Spanish (es)

Italian (it)

Japanese (jp)

Korean (ko)

Simplified Chinese (zh-chs)

Traditional Chinese (zh-cht)

These can be downloaded from the Business Objects Web Site.

Custom language resource files, which you create as text strings and then compile as custom language resource DLLs.

Neutral and Non-neutral Resources

Language resources include both neutral and non-neutral resources.

Neutral resources represent a general language setting, such as en (for English), fr (for French), and jp (for Japanese).

Non-neutral resources represent a version of the language that is qualified based on an additional criterion, such as region: for example, de-AT for German (Austria).

Global and Local Resources

Default language resources are installed in the GAC and therefore are always available globally, but custom language resources may be accessed either globally or locally.

The custom language resources folder may be stored in a central location and shared from that location, or it may be copied to the local directory of the Web or Windows application.

When the custom language resources are stored in a central location, the resources are referred to as global resources. In that scenario, the Web or Windows application must be informed of their location by settings in an XML configuration file. For a global resource to load correctly, the name of the folder that contains the resource files must match the language locale.

If the custom language resources are copied to the local directory, the resources are referred to as local resources. In that scenario, no configuration file is necessary. For a local resource to load correctly, the culture of the resource files and the name of the folder that contains the resource files must match the language locale.

Choosing to Use Global or Local Resources

Global resources are the preferred choice when resources must be shared with more than one application. For example a Web server, with multiple Web sites that require a common set of language resources to be shared across all Web sites, is best suited to use global resources.

Local resources are preferred when resources are used by a single application, whose wide distribution mandates a simple configuration. For example a Windows application, which consists of a single application folder that is intended for wide distribution, is best suited to use local resources.

In most cases, Web applications are best suited for global resources, and Windows applications are best suited for local resources.

Which resource loads first?

If you configure both local and global resources for a particular application, the local resources take higher priority (they come first in the load order). If you want to guarantee that global resources are used, verify that your configuration file is correct, and remove the local resources folder.

Non-neutral and neutral resources are also affected by the resource loading order. If you have configured both non-neutral and neutral resources for a particular language (such as, "German (Austria)" and "German"), the non-neutral resource is checked first.

In both cases, the specific takes precedence over the general.

Client-side display

The following factors determine the language that the client application displays at runtime:

In a Windows application:

- The Environment Locale settings on each user's machine.

In a Web Site:

- The Environment Locale settings of the Web server.
- The Language settings of the client browser.
- The Culture property, which you set on a separate ASPX page.

Useful Addresses at a Glance

Address	Content
Business Objects Product Information http://www.businessobjects.com	Information about the full range of Business Objects products.
Product Documentation http://www.businessobjects.com/support	Business Objects product documentation, including the Business Objects Documentation Roadmap.
Business Objects Documentation Mailbox crdocs@businessobjects.com	Send us feedback or questions about documentation.
Online Customer Support http://www.businessobjects.com/support	Information on Customer Support programs, as well as links to technical articles, downloads, and online forums.
Business Objects Consulting Services http://www.businessobjects.com/services/consulting	Information on how Business Objects can help maximize your business intelligence investment.
Business Objects Education Services http://www.businessobjects.com/services/training	Information on Business Objects training options and modules.

Which Persistence Approach Should I Use with Crystal Reports?

When planning a Web application to build with the Crystal Reports SDK, one of your most important considerations is which persistence approach to use. Learning the SDK fundamentals that affect persistence helps you choose the best structure for your Crystal Reports for Visual Studio 2005 project.

What is persistence?

Web pages do not preserve state (the status and information of a user connecting to a website). Each Web page is requested from a server, sent to the user, and the process is then terminated. This is fine when reading text information, but problematic when designing a Web application that needs to preserve information about a user across page reloads and redirects.

Persistence refers to the use of a mechanism to preserve state for each user (such as the current report page being viewed by each user) unhindered by page reloads and redirects.

Best practices for persistence

In this section, you explore best practices for persisting changes made to a Crystal report on a Web page while that report is refreshed during a Web page reload. The change that needs to be persisted may be as simple as going to the second page after the report viewer's Next Page button has been clicked, or as complex as displaying entirely different data after a report's parameters have been changed.

In Crystal Reports for Visual Studio 2005, persistence must be applied to the following things:

The CrystalReportViewer control.

The report that is bound to the CrystalReportViewer control.

Ways of persisting state in ASP and ASP.NET

In traditional ASP and ASP.NET, the state is maintained by either the Web browser or the Web server, in the following ways.

Environment	Client or server?	Name	Method
ASP*	Client browser	Form fields	Pass name/value pairs across Web pages through form submission fields.
ASP*	Client browser	URL arguments	Pass name/value pairs across Web pages using URL arguments.
ASP*	Client browser	Cookie	Assign name/value pairs to a cookie on the client browser, and then retrieve the cookie in a new page.
ASP*	Web server	Session object	Assign instantiated objects to the Session object on the

			server, and then retrieve from Session in a new page.
ASP*	Web server	Application object	Assign instantiated objects to the Application object on the server, and then retrieve from Application in a new page.
ASP.NET	Client browser	ViewState object	Assign string values to the ViewState object of the ASP.NET Web Form.
ASP.NET	Web server	Cache object	Same as Application object, but with enhanced features.

* All ASP persistence approaches continue to work in ASP.NET.

Which persistence approaches work best with Crystal Reports?

The most appropriate persistence approaches to use with Crystal Reports are ViewState, Session, or Cache.

The approach you choose to use depends on the object model:

ViewState and Persistence of the CrystalReportViewer Object Model.

Session and Persistence of the ReportDocument Object Model.

Cache and Persistence of the ReportDocument Object Model.

ViewState and Persistence of the CrystalReportViewer Object Model

What is ViewState?

ViewState is a browser-based approach in ASP.NET for persisting the state of the view, that is, the Web Form. Its primary function is to support the persistence of Web controls.

Web controls (also called Web server controls) are modeled on Windows controls, which were introduced in Visual Basic. Windows controls are objects on the form that encapsulate a piece of display functionality, such as a text field, a button, or a table of data.

Web controls are similar to Windows controls. Like Windows controls, they operate on two levels: within the Web page, and in the code-behind class that supports the Web page. Similar to traditional controls in Windows Forms, Web controls encapsulate discrete pieces of display functionality into GUI objects: Button, TextField, DropDownList, DataGrid, and so on. In the code-behind class, these same controls are paralleled as classes that expose properties and methods.

A Web page is different from a Windows form in that a Web page is a stateless environment. Therefore some kind of persistence mechanism is required to preserve the state of the Web page across page reloads.

ViewState maintains the state of controls on the Web page, much as Session maintains the state of instantiated objects on the server.

Note ViewState maintains the state of all Web controls automatically. This is achieved by having ViewState store each control based on the control's EnableViewState property (which defaults to True).

Because ViewState stores the state of the data from the Web controls on the page, the entire ViewState object must be enclosed within the page as the page is transferred back and forth between the browser and the Web server. This is done by encrypting the entire ViewState object as a string, and then placing this string within the value of a hidden form tag on the page. For example, the ViewState of an ASP.NET Web page that contains only a single button control has the following HTML code:

```
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTMwNzcxMzI0Ozs+I7GfLyg3p44eTLFCiVEiRKUBzFw=" />
```

ViewState stores only information that can be converted to string format.

Persisting the report display of the CrystalReportViewer control

The CrystalReportViewer control performs the role of report display for a Crystal report. It renders the report into html on the page along with a toolbar and tree view for manipulating the report display. The toolbar contains buttons to zoom, go to next page, print, export, and so on. The tree view expands to show nested grouping of data.

ViewState persists control information; therefore it persists the state of all report display information (including toolbar and tree view events) for the CrystalReportViewer control across page reloads.

For example, if a user were viewing page 3 of the report and clicked the next page button in the toolbar of the CrystalReportViewer control, ViewState would persist the state of both pieces of information:

The current page number.

The state of the Next Page button (clicked).

During page reload, the ViewState would restore the CrystalReportViewer control to page 3, and then restore the next page event click, causing the control to move the report ahead to page 4.

Persisting the object model of the CrystalReportViewer control

The CrystalReportViewer control performs an additional role: not only report display, but also a limited object model (contained in the CrystalReportViewer control class). This limited object model can be used for programmatic interaction with the report.

ViewState persists the state of both roles:

The report display.

The CrystalReportViewer object model.

However, use of the CrystalReportViewer object model is generally discouraged, in favor of the more extensive ReportDocument object model. This alternate object model is not contained within the control but is part of the class libraries in the SDK.

Sharing persistence mechanisms

If you use the CrystalReportViewer control to perform both roles (report display and object model), ViewState persists both of them, and you do not need any additional persistence mechanisms.

> **Note** An example of this would be binding the CrystalReportViewer control to a file directory path.

However, if you choose to use the CrystalReportViewer control only for the role of report display, and then bind the control to an external object model (such as ReportDocument), a separate persistence mechanism is required to persist that external object model. Typically that second persistence mechanism is Session (or occasionally, Cache).

Session and Persistence of the ReportDocument Object Model

What is Session?

Session is a Web server-based approach used in both ASP and ASP.NET for preserving state. Session allows you to persist any object for the entire length of a user's session by storing that object in the Web server's memory.

Session is typically used to do either of these things:

- Store information that needs to persist its state during the entire length of a user session, such as logon or other information required as users navigate the Web application.
- Store an object that needs to persist its state only across a page reload or a functionally grouped set of pages.

The strength of Session is that it preserves the user's state information on the Web server for access at any time from any page. Since the browser doesn't have to store any of this information, any browser can be used, even browser devices such as PDAs or cell phones.

Limitations of this persistence approach

- The amount of server memory required by Session grows as more users log on.
- Each user who accesses the Web application generates a separate Session object. Each Session object lasts for the length of the user's visit plus an inactivity period.
- If many objects are persisted into each Session, and many users are using the Web application simultaneously (creating many Sessions), the amount of server memory devoted to Session persistence can become significant, limiting scalability.

For alternate persistence approaches see the following:

- ViewState and Persistence of the CrystalReportViewer Object Model
- Cache and Persistence of the ReportDocument Object Model

Persisting the ReportDocument object model with Session

If the report has been encapsulated within the ReportDocument object model, the ReportDocument object model must be persisted using a server-based approach such as Session or Cache.

To persist a report within the ReportDocument object model using Session, instantiate the ReportDocument and then assign it to Session.

Session is the simplest approach and is preferred when you are learning to build an ASP.NET Web application using Crystal Reports. It is also the recommended approach for storing ReportDocument instances where the report has low shareability.

Limitations of persisting the ReportDocument object model with Session

Whenever a ReportDocument instance has a high degree of shareability, consider using Cache instead of Session.

Contrasting Session and ViewState

Session is primarily concerned with persisting the state of objects in the code-behind class. ViewState is primarily concerned with persisting the state of controls on the Web page. When a control on the Web page is bound to an object in the code-behind class and both need to be persisted across page reloads, Session and ViewState share the roles of persistence.

In this case ViewState persists a CrystalReportViewer control, and Session persists a ReportDocument object that is bound to the control.

Cache and Persistence of the ReportDocument Object Model

What is Cache?

Cache is a server-based approach in ASP.NET for preserving state. Cache is functionally similar to the Application object found in both ASP and ASP.NET:

Application allows you to persist any object across application scope.

Objects placed into Application are available to all users. But Application, being general to the entire application, is not designed to preserve user-specific information.

Cache shares all of these features with Application, but it adds new levels of intelligence for managing transient data:

An object added to the cache can be configured with file-based, key-based, or time-based dependencies. If the associated file or key changes, or if a certain period of time passes, the object is automatically removed from the cache, and an updated version is placed in the cache the next time the object is required.

An object added to the cache that has no dependencies and is underused is automatically expired.

When an object is removed from cache, an event is triggered. You can write code to run on that event and load an updated version of the object to the Cache.

If an alternative version of an object is added using the original key string, it overwrites the previous version. To prevent overwriting, concatenate the alternative definition to the key string to make each alternative version of the object unique.

The strength of Cache over Application is that, like Application, it stores information that is accessible to all users, but Cache can also update itself based on changes in its dependencies.

Limitations of this persistence approach

Developers who are new to Cache may be tempted to use it everywhere for persistence, replacing Session with Cache. However, Cache is not designed to replace the functionality of the Session object. Attempting to emulate Session uniqueness by concatenating user-specific data to the Cache key loads up the Cache with user objects that, unlike with

Session, does not expire after a user's timeout period. As a result, Cache ends up placing a higher demand on Web server memory than the Session object did.

If you need to persist user-specific data, continue to use the Session object.

For alternate persistence approaches see:

ViewState and Persistence of the CrystalReportViewer Object Model

Session and Persistence of the ReportDocument Object Model

Persisting the ReportDocument object model with Cache

If the report has been encapsulated within the ReportDocument object model, the ReportDocument object model must be persisted using a server-based approach such as Session or Cache.

You can persist a report within the ReportDocument object model using Cache in one of two ways:

Instantiate the report and then assign it to the Cache object, using the same syntax as for assigning a report to the Session object.

This method only works for a report that has high shareability, where the ReportDocument instance occurs exactly once, using a single set of parameters and logon credentials. For a ReportDocument instance that may occur multiple times due to variation in its parameters and logon information, reassigning to the Cache object using the same key string overwrites the previous version of the ReportDocument instance.

Instantiate a version of the report class that implements the ICachedReport interface.

The Crystal Reports SDK includes a built-in caching framework for reports. Any report that implements the ICachedReport interface is automatically added to the cache with a unique key based on parameters and user logon credentials. This method works for any report that has high shareability, but may have a few versions due to minor variations in parameters and logon credentials. For reports with low shareability (that are user-specific), assign them to the Session object instead.

To persist an embedded report that implements ICachedReport

1. Add the report to the project.

   This creates an embedded report class. It also creates a cached report class that loads and returns a cached instance of the embedded report class.
2. Instantiate the cached report class.
3. Assign the cached class instance to the CrystalReportViewer control.

To persist a non-embedded report through a utility class that implements ICachedReport

1. Create your own cache management utility class and set it to implement ICachedReport.
2. In this utility class, load the non-embedded report from a path string using the ReportDocument.Load() method.
3. Code the implementation method CreateReport() to return the ReportDocument instance of the non-embedded report.
4. Instantiate the report cache management utility class.
5. Assign that class instance to the CrystalReportViewer control.

Limitations of persisting the ReportDocument object model with Cache

Cache is the best approach to use when persisting ReportDocument instances that have a high degree of shareability across users. If the report is user-specific, then Cache will waste server memory creating user-based instances at an application level that will last in server memory past user expiry. User-specific reports should be assigned to Session instead.

> **Note** In most cases, use Session to persist ReportDocument instances. Use Cache (or more specifically, the ICachedReport interface) only when a report has high shareability and when the report is very large, or so complex that it takes several minutes to retrieve its data.

Contrasting Cache and ViewState

Cache is primarily concerned with persisting the state of objects in the code-behind class. ViewState is primarily concerned with persisting the state of controls on the Web page. When a control on the Web page is bound to an object in the code-behind class and both need to be persisted across page reloads, Cache and ViewState share the roles of persistence.

In this case ViewState persists a CrystalReportViewer control, and Cache persists a ReportDocument object that is bound to the control.

Persistence Limitations When Report Binding in the Page_Load Event Handler

In ASP.NET Web applications it is common practice to place all startup code for the page in the Page_Load event handler, which is called by the Page.Load event.

In particular, control data binding code is usually stored within the Page_Load event handler. However, placing the binding code in this event handler causes a problem with ViewState. The problem and typical resolution is as follows:

> ViewState is used to persist two things across page reloads: the data that is bound to the control, and mouse clicks events performed on the control.
>
> ViewState is a string. Therefore both the data and click events must be serialized.
>
> During page reload, both the data and click events are restored from ViewState.
>
> The Page.Load event occurs after ViewState has been restored. If the Page_Load event handler contains control binding code, at time of page reload this binding code will overwrite ViewState, losing both the original data and the click events.
>
> This problem is typically experienced as controls that forget mouse click actions (e.g. a DropDownList selection) when the page reloads.
>
> To prevent data and mouse click events from being overwritten, any binding code in the Page_Load event handler is placed within a Not IsPostBack conditional block, which prevents the binding code from being called during postbacks.

This resolution makes one crucial assumption: that both the data and mouse click events can be serialized to ViewState. However, the CrystalReportViewer control binds to objects that are non-serializable (specifically, the ReportDocument class, the ReportClientDocument class or the InfoObject class.)

> **Note** There is one exception: when the CrystalReportViewer control is bound to a report by its file directory path, the path string can be persisted to ViewState. In this

scenario only, the CrystalReportViewer control could be placed in a Not IsPostBack conditional block. However, this report binding scenario is less powerful and less commonly used than binding to the report classes listed above.

Since only the CrystalReportViewer control's mouse click events can be serialized to ViewState, binding to a non-serializable report class creates an irresolvable problem during page reloads:

If the report binding code is placed within a Not IsPostBack conditional block, then the mouse click events from ViewState will be preserved. However, report binding does not occur. Therefore, an exception is thrown.

If the report binding code is placed outside the conditional block, the report is bound correctly. However, ViewState is overwritten in the process. Therefore, the mouse click events are lost.

Note This approach is typically observed when clicking through a multiple-page report in the CrystalReportViewer control. The report mysteriously keeps returning to page 1.

Recommended solution: move the CrystalReportViewer control's binding code to the Init event

The solution for the CrystalReportViewer control is to move your report binding code to the Init event, which occurs before ViewState is restored.

This solution raises one complication. Since the Init event is coded against less commonly than the Load event, it is harder to access. In Visual Studio .NET 2002 or 2003 Web or Windows projects, the Init event handling code is located within the Web Form Designer Generated code region, an area that is typically hidden and reserved for generated code.

To resolve this, the following approach is recommended:

Extract all CrystalReportViewer binding and configuration code into a private helper method named `ConfigureCrystalReports()`.

Within the Web Form Designer Generated code region, place only a single line of code in the `Page_Init()` event handler or `OnInit()` event raising method: a call to the `ConfigureCrystalReports()` helper method.

Instructions for creating and populating the `ConfigureCrystalReports()` helper method are given in Project Setup.

.NET Framework 2.0

To deploy Visual Studio 2005 applications, the .NET Framework 2.0 must be installed on the target machine, before you install the deployment project. When you create a deployment project, the .NET Framework 2.0 is not included by default.

You can find the .NET Framework on the Visual Studio .NET Windows Components Update CD or from the Microsoft Web Site. From the CD, the .NET Framework installer (an .exe file), can be redistributed with the deployment project.

For ClickOnce deployment, you can include the .NET Framework 2.0 as a prerequisite setup to the published Web site.

Crystal Reports for .NET Framework 2.0 Windows Installer

You can use Windows Installer in place of merge modules for deployment projects. The Windows Installer allows for smaller deployment projects and reduced installation time.

When you create an installer for an application that uses Crystal Reports for Visual Studio, a setup package is made. This setup package installs Crystal Reports for .NET Framework 2.0 runtime files on target machines. It creates two setup files: an .exe and an .msi. In the .exe setup file, merge modules are included in a deployment project because they install the Crystal Reports runtime. However, if the Crystal Reports for .NET Framework 2.0 Windows Installer .msi file is used for installation on target machines, merge modules do not need to be added.

You can use the Windows Installer when many deployment projects are installed on the same target computer. In this case, Crystal Reports runtime files are installed once. If merge modules are used, the Crystal Reports runtime is installed each time a deployment project is installed on the target computer.

In Crystal Reports 10 and Visual Studio .NET 2002 or 2003, the Windows Installer includes CrystalReports10_NET_EmbeddedReporting.msm and CrystalReports10_maps.msm merge modules. All database drivers and map objects are installed when you run the Windows Installer. Merge modules provide the flexibility to choose whether the mapping merge module is required and to decide which database driver to include in the deployment project.

Note In Crystal Reports for Visual Studio 2005, map objects are not supported; therefore, there is no mapping merge module.

Merge Modules Summary

The merge modules that are used in the deployment process vary across the different versions of Crystal Reports and Visual Studio. The following table outlines the merge modules that are needed for each software configuration.

Note The addition of more merge modules than specified in the scenarios may cause deployment problems.

Save the merge module files to your machine in C:\Program Files\Common Files\Merge Modules.

Software	Merge Modules	File name
Crystal Reports 9	reportengine.msm crnetruntime.msm mapping.msm license.msm	cr9netmergemodules.zip
Crystal Reports 10	CrystalReports10_NET_EmbeddedReporting.msm CrystalReports10_NET_RemoteReporting.msm CrystalReports10_NET_WebServiceReporting.msm	cr10_net_mergemodules.zip
Crystal Reports 11	crystal11_net_embeddedreporting.msm CrystalReports11_maps.msm	Automatically installed to C:\Program Files\Common Files\Merge Modules
Crystal Reports for Visual Studio .NET 2002	Database_Access.msm Database_Access_enu.msm Managed.msm regwiz.msm	cr_net_mergemodules_en.zip
Crystal Reports for Visual Studio .NET 2003	Crystal_Database_Access2003.msm Crystal_Database_Access2003_enu.msm Crystal_Managed2003.msm Crystal_regwiz2003.msm	cr_net_2003_mergemodules_en.zip
Crystal Reports for Visual Studio 2005	Depends on the machine type and project type.	See http://www.businessobject

	See Crystal Reports Merge Modules for Visual Studio 2005.	s.com/products/dev_zone/net/2005.asp to download the merge modules used in Crystal Reports for Visual Studio.

To locate and download a merge module from the Business Objects technical support Web Site, search for the merge module .zip file name:

http://support.businessobjects.com/search/

Crystal Reports Merge Modules for Visual Studio 2005

For the specified machine and project type, you must add the correct merge modules to the project.

Note In Crystal Reports for Visual Studio 2005, map objects are not supported; therefore no mapping merge module exists.

The merge modules that are used for Crystal Reports for Visual Studio 2005 can be downloaded from http://www.businessobjects.com/products/dev_zone/net/2005.asp.

Machine type	Project type	Merge modules required
32-bit	Windows applications/ Web Sites	CrystalReportsRedist2005_X86.msm
x64 (AMD64)	Windows applications/ Web Sites	CrystalReportsRedist2005_X64.msm
IA64 (Itanium)	Windows applications/ Web Sites	CrystalReportsRedist2005_IA64.msm