SAMPLE CHAPTER

SQL SERVER 2005
Reporting Services
IN ACTION

Revised Edition of
Microsoft Reporting Services in Action

Bret Updegraff
Foreword by Brian Welcker

MANNING

SQL Server 2005
Reporting Services in Action
by Bret Updegraff
Sample Chapter 11

brief contents

CHAPTER 11

Mastering the ReportViewer controls

ReportViewer controls are part of Visual Studio 2005 and, when used, are guaranteed to change the way you integrate RS reports into your applications. The ReportViewer controls are built into the toolbox of Visual Studio 2005 and don't require any downloads or additional installation. When you want to place an RS report in your windows or web application, you can simply drag the control into the design environment, configure the control with properties for your report, and run your application. It's that easy. Why dedicate an entire chapter to these controls if it's that easy, you may ask? Let's just say it *can* be that easy, but as with any of the .NET controls, there are many ways to configure and tweak them to meet your needs.

In this chapter you learn how these controls work. We examine the different modes of these controls, and explain when you'll want to use each mode. You also see how these controls can be used to integrate the reports that have been deployed to the Report Server (in remote mode), as well as how these controls can be used to render reports into your applications without the use of a Report Server (in local mode). In

addition, we explore how to implement custom validation of your report parameters and how to convert your Report Server reports so that they can be used in an environment that is disconnected from a Report Server. Finally, we discuss deploying applications that use the ReportViewer controls and what you need to do for your deployments to be successful.

Let's get started by learning how these controls work.

11.1 How the .NET ReportViewer controls work

In chapter 10 we covered the architecture for pulling reports out of the Report Server. While that information is very important for the report developer, .NET application developers won't need this level of understanding to add reports to their applications using the ReportViewer controls. Because the ReportViewer controls mask the RS architecture from application developers, developers can add existing reports to their applications without having a strong background in Reporting Services.

Using these controls should be your method of choice when integrating RS reports into your Visual Studio 2005 Windows and web applications for two reasons:

- Ease of use
- The ability to completely manage the report properties through these controls

Using the ReportViewer controls allows you to spend more time focusing on the business logic of your applications and less time integrating reports into your applications.

Let's look first at how the ReportViewer controls work for web applications versus Windows applications.

11.1.1 Controls for web and Windows applications

There is a ReportViewer control for both Windows and web applications. Depending on the type of application you are working with, the Visual Studio toolbox will contain the appropriate control. While at first glance these controls seem to be identical, there are actually a few subtle differences between them. Table 11.1 showcases these differences.

Table 11.1 Differences between the Windows and web ReportViewer controls

ReportViewer control feature	Web	Windows
Presentation	Uses HTML formatting to display a report.	Uses a Graphical Device Interface (GDI) to provide a visual experience that is consistent with Windows user interface styles.
Processing	Local report processing can be configured for asynchronous processing.	Local report processing is always performed as an asynchronous process.

continued on next page

Table 11.1 Differences between the Windows and web ReportViewer controls *(continued)*

ReportViewer control feature	Web	Windows
Printing	Printing reports from the web server control uses an ActiveX print control if the report is processed on a remote server. If you want to print a locally processed report from the web server control, you can export the report to another output format before you print.	Printing reports from the Windows Forms control uses the print functionality of the operating system.
Deployment	The deployment strategy for reports hosted in the Web server control in an ASP.NET application must take session state and web farm configuration into account. If you are using the web server control to process a report on a remote Report Server, you must consider how to authenticate application users to access the server and any external data sources that provide data to reports.	If you are deploying the ReportViewer with a Windows application, you will need to use the bootstrap features in Visual Studio to be sure that ReportViewer is installed on the client machine along with your application. See section 11.6 of this chapter for more information.

The ReportViewer controls each have two processing modes: remote and local. Let's take a closer look at these modes so that you'll know when to use each mode and also understand the differences as well as limitations of each.

11.1.2 Choosing remote or local mode

The ReportViewer controls can process and render a report with or without Report Server access. If you have server access (and this is more common), you choose *remote* mode. Remote mode allows the application developer to select a Report Server and path to a report as well as control most of the properties and features of RS by changing properties of this control. We explore these properties in section 11.1.3.

In some cases, you might not want to access the Report Server, or you may not have access to it, so you choose *local* mode. Local mode provides some additional options that take Reporting Services to the next level from its original version. You are no longer tied to the Report Server, which means that you can create reports that run from local data sources in a disconnected environment. Local mode also enables you to use objects and Web services as data sources for your reports. This opens a whole new world of reporting for you.

Regardless of the processing mode, ReportViewer-generated reports look and function in a very similar way. To help you better appreciate the differences between processing modes, table 11.2 lists several ReportViewer features and how these features are affected by the mode used.

Table 11.2 Differences between remote and local modes

ReportViewer control feature	Remote mode	Local mode
RDL Management	Report definition (RDL) is supplied and rendered by the Report Server.	The RDL is supplied by the host application instead of being retrieved from a Report Server.
Report Engine	Uses the Report Server engine.	Uses the same engine as the Report Server but is embedded into the application.
DataSet	Supplies the data as SQL Server DataSet.	Supplies the data as an ADO.NET DataTable to the report engine.
Export Formats	Full Export options (see chapter 1).	Only exports to Excel or PDF format.
Report Creation	Report Creation is done in the Business Intelligence Development Studio with the Reporting Services project.	Report creation is integrated into the Windows application VS 2005 project instead of having a separate Reporting Services project.

Before you see each processing mode in action, let's learn about some of the ReportViewer properties that you'll encounter most often.

11.1.3 Managing properties of the ReportViewer controls

Both modes of the ReportViewer controls share a number of common properties that can be set. For example, you might want to change the color for the links in the toolbar (`LinkActiveColor` and `LinkActiveHoverColor`) or the height and width of the ReportViewer (`Height` and `Width`). Another common configuration is to hide or show the toolbar itself by using the `ShowToolbar` property. We explore this in more detail in section 11.2.2.

Table 11.3 lists the properties you will use most of the time. Two of the properties that we cover in our examples later in this chapter are the `ShowToolbar` and the `ShowContextMenu` properties.

Table 11.3 does not contain the complete list of properties; for that, see the SQL Server Books Online documentation or search for "ReportViewer Controls (Visual Studio)" in the Visual Studio product documentation. The best way to understand how these properties affect the ReportViewer controls is through example. In the next section, we walk you through an example of using the ReportViewer in remote mode, and we also configure many of the properties that are shown in table 11.3.

Table 11.3 Commonly used properties of the ReportViewer controls

Member	Description
`ProcessingMode`	Gets or sets the processing mode for the control. Possible values are `Remote` and `Local`.
`DocumentMapWidth`	Gets or sets the document map width in pixels.
`LinkActiveColor`	Gets or sets the active color for links in the toolbar. Note that this does not have any effect on links in your reports.
`LinkActiveHoverColor`	Gets or sets the hover color for links in the toolbar. Note that this does not have any effect on any links in your reports.
`LinkDisabledColor`	Gets or sets the color for disabled links in the toolbar. Note that this does not have any effect on any links in your reports.
`PromptAreaCollapsed`	Gets or sets a Boolean value that determines whether the parameter area is initially collapsed or expanded. You must specify a default value for all parameters if you are going to set this to true. If you do not specify default values, you will receive an error similar to the following: *The 'Customer Name' parameter is missing a value*.
`ShowParameterPrompts`	Gets or sets a Boolean value that determines whether the parameter area is shown in the control. You must specify a default value for all parameters if you are going to set this property to false. If you do not specify default values, you will receive an error similar to the following: *The 'Customer Name' parameter is missing a value*.
`ShowToolbar`	Gets or sets a Boolean value that enables or disables the HTML Viewer toolbar. If you are disabling the toolbar as a means to limit the end user's functionality for Windows applications, you should also set the `ShowContextMenu` to false.
`ShowContextMenu`	Gets or sets a Boolean value that enables or disables the context menu for the ReportViewer control. This property is only available in the Windows ReportViewer control. The context menu will expose many of the features in the toolbar such as print, export, zoom, and page properties.
`ShowProgress`	Gets or sets a Boolean value that determines whether a progress animation is shown while waiting for the report to render.
`Height` and `Width`	These properties allow you to get or set the `height` and `width` properties of the ReportViewer control. For the Windows control you need to specify this value as number of pixels (without adding px at the end of the number). For the web control you can use both pixels or percentages for the width column. While the `width` property works well with both pixels and percentages, the `height` property does not seem to function well with percentages.

11.2 USING REPORTVIEWER IN REMOTE MODE

As we stated earlier, using the ReportViewer controls in remote mode forces you to pull the reports directly from the Report Server. This means that the reports have already been created and deployed to the Report Server and you use the ReportViewer controls in remote mode to get the reports and render them into your applications.

In this section, you see some examples of using the ReportViewer control to pull data from objects. Also, we cover some of the many properties of the ReportViewer controls and explore some real-world examples. For these walkthrough examples, we encourage you to create your own Windows or web projects and use the code provided with the book as a reference. You'll find the code samples in the chapter 11 folder in the AWReporterWin and AWReporterWeb projects.

11.2.1 Creating, configuring, and running the control

In this section you learn how to create and configure the ReportViewer controls in remote mode by doing the following:

- Create a Windows Form and add the ReportViewer to it.
- Configure the properties of the ReportViewer.
- Run the Windows Form and view the Sales By Territory report from part 1 in the ReportViewer.

Adding the ReportViewer control to a Windows Form

The example code for this section is in the chapter 11 folder of the AWReporterWin project. We recommend you create your own Windows application project for this walkthrough. Once you have a project to work with, follow these steps:

Step 1 Create a Windows Form project and name it **ReportViewerRemote.cs**. If you created a new project for this walkthrough, you can use the default form in the project named Form1.cs and rename it to **ReportViewerRemote.cs**. You can find the ReportViewer control within the data controls of the toolbox (see figure 11.1).

Step 2 Drag and drop this control onto the Windows Form.

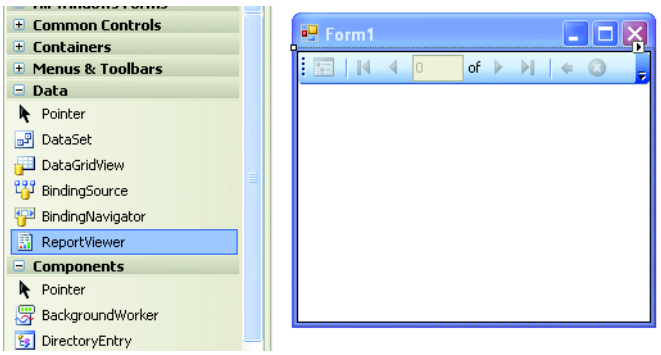

Figure 11.1 The ReportViewer Control is shown here in a Windows Form.

Configuring the ReportViewer control

Now you need to enter all of the required information to integrate an RS report into a Windows application. Follow these steps:

Step 1 Set the size properties and dock the control to the form. For this example you want to set the size properties to about 600 pixels wide by 450 pixels high. When the ReportViewer control is dropped onto the form, the smart tag window appears. Dock the ReportViewer control to your form by clicking on the *dock to parent form…* link in the smart tag window. If you don't see the smart tag window, you can get to it by selecting the ReportViewer control and clicking on the small arrow icon, as shown in figure 11.1.

Step 2 Set the control properties. First, expose the properties by selecting the control and viewing the properties window for the control. Let's set the `ShowZoomControl` property to false, as shown in figure 11.2. You can see that there are a number of items in the toolbar that can be shown or hidden when the report is rendered. We work with these properties a little later on, but for now let's switch our attention to configuring the control to process the Sales By Territory Interactive report that you created in chapter 4.

Step 3 Configure the control. To view configuration properties, click on the smart tag icon found in the upper right of the control. Since we are exploring the remote mode of the ReportViewer, select `<Server Report>` from the `Choose Report` field.

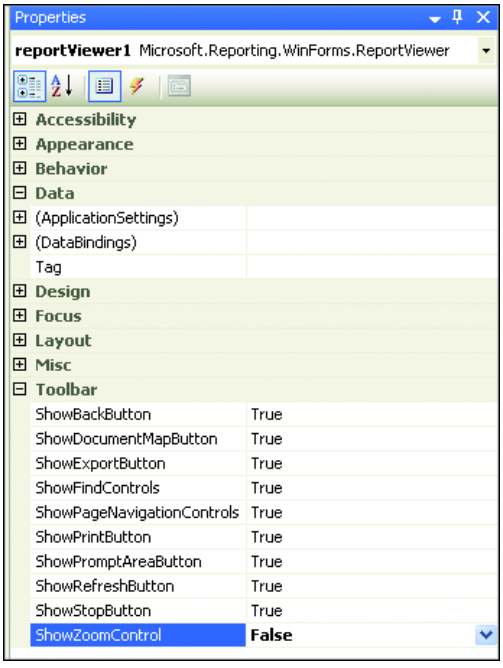

Figure 11.2
You can hide or show toolbar buttons by modifying properties of the ReportViewer control.

NOTE Choosing `<Server Report>` automatically sets the `Processing Mode` property to `Remote`. We could have simply gone through the property settings to configure this control, but the smart tags settings are an easier way to accomplish our goal.

Step 4 Enter a valid Report Server URL and report path (shown in figure 11.3). Be sure to include the full path for the report starting with a slash (/). Also be sure to leave the extension off the report name. For this example, the report path should be **/AWReporter/Sales By Territory**.

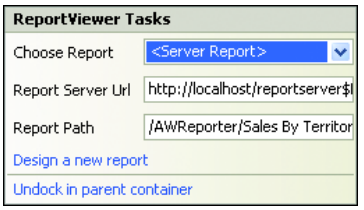

Figure 11.3 The smart tag window exposes a great starting point for configuring the ReportViewer control.

Running and viewing your report

You are now ready to see the results of your work. When you run the application, you should see the report shown in figure 11.4.

Let's take a look at how you might customize the ReportViewer.

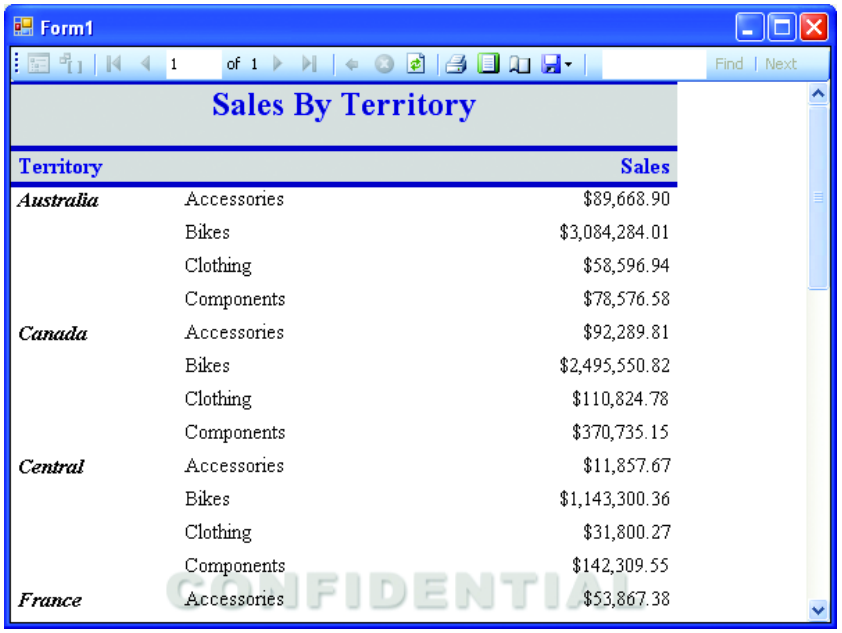

Figure 11.4 In the Sales By Territory report, the Zoom functionality is hidden from the toolbar.

11.2.2 Additional customizations for the ReportViewer control

Let's say you don't want your end users to see the HTML toolbar. Maybe you have your own toolbar in your application, or maybe you just don't want your users to have the functionality provided in the HTML toolbar. Simply set the `ShowToolbar` property to false from the ReportViewer control property window (figure 11.2). The toolbar is shown right above the report title in figure 11.4.

If you are trying to disable functionality from the end user, hiding the toolbar won't enforce this. Users can simply right-click on the rendered report to show the context menu, which exposes the toolbar functionality. To truly disable all toolbox functionality, you need to either set both `ShowToolbar` and `ShowContextMenu` to false or set the properties for the functionality that you want to disable.

What if you want to disable some of the functionality, but not all of it? In this hypothetical situation, you don't want your users to be able to export or print this particular report, but you do want them to have the ability to zoom in and out. Therefore, you need to expose the toolbar, but to disable the export or print functionality, you have to set `ShowExportButton` and `ShowPrintButton` to false. Not only will the control hide the buttons, but if users right-click on the control they won't see the option to export or print.

Using remote mode allows you to add the reports that you created and deployed to your Report Server in part 1 of this book. Using this mode is the easiest and quickest way to add existing reports to your applications. After reading this section, you should be armed with the knowledge needed to integrate, configure, and customize existing reports into your .NET 2.0 applications.

NOTE Even though there are separate controls for Windows applications versus web applications, it is important to understand that the differences are minor. If we had demonstrated this in an ASP.NET web application, you would have seen that all of the steps are virtually the same.

In the next section, we examine the other side of the ReportViewer controls: local mode.

11.3 USING REPORTVIEWER IN LOCAL MODE

The local mode of the ReportViewer controls provides rich reporting without the use of a Report Server by embedding the report definition inside the application. This is a great way to use Reporting Services when you can't access a Report Server.

To use local mode, you first have to create a report using Visual Studio 2005 from within your Windows or web application project. This is different from remote mode, in which you used reports that were created using the Report Designer and deployed to a Report Server. This means that with local mode you cannot natively use reports that you have already created and deployed to your Report Server.

Report Server (remote mode) reports have an `.rdl` extension whereas local reports used by the ReportViewer controls use the `.rdlc` extension. Later in section 11.5.1 you learn how to convert your RDL files for use with local mode.

In this section, you learn how to create the RDLC from your Windows or web applications by creating local reports from a variety of data sources. Let's start off by using the ReportViewer control to get information from a database.

As before, for this walkthrough example you can reference the sample code in the Chapter 11 folder of the AWReporterWin project.

11.3.1 Creating a local report with a database as the data source

As stated earlier, in local mode the report file is created as an RDLC file instead of an RDL file. A second difference is that the report is created from within your application projects instead of a separate Reporting Services project. For the most part, everything else will feel the same as it did with remote mode, but there are some minor differences. We showcase many of these differences in our examples.

For this hypothetical situation, you've been asked to make the AWC employee directory available to field sales agents who don't typically have access to the AWC network. Let's assume that there is a process built that replicates or syncs the data from the AWC database to a database on the sales agent's local machine.

> **NOTE** We chose to run this from the same local database that the other examples run from. In a real-world scenario, the client computer would likely be running SQL Express and would also not have an exact replica of the original database. To simplify the code setup for this book, we simply pretend this is a separate database.

Step 1 Create a new Windows Form and call it **ReportViewerLocal.cs**. Stretch the form, add the ReportViewer, and anchor the control just as you did in the ReportViewer remote example earlier. Instead of choosing `<Server Report>` from the `Choose Report` property of the smart tag window (figure 11.3), click the *design new report* link. This creates a new RDLC file and opens it up in your project. On the left side of your design window you should see a Data Sources tab; if not, you can add it by selecting Data > Show Data Sources from the top menu. You can also toggle the Data Sources view by pressing Shift-Alt-D.

Step 2 Create a data source for your report by right-clicking in the Data Sources section or by selecting Data > Add New Data Source from the top menu. This opens the Data Source Configuration Wizard, as shown in figure 11.5.

Step 3 Select Database and click the Next button. The next screen lets you choose your data connection by either selecting an existing connection or creating a new connection. Let's set this data source up for our AdventureWorks database.

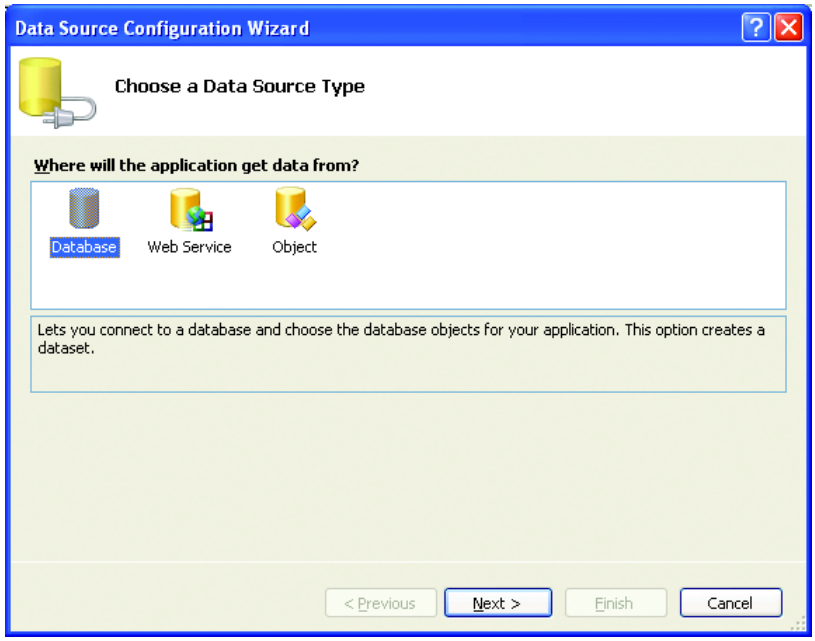

Figure 11.5 The Data Source Configuration Wizard allows you to create data sources from a database, Web service, or an object.

Step 4 Once you've set up the data connection and clicked the Next button, you'll see a screen that prompts you to name your dataset and choose which database objects you want to include. Figure 11.6 shows the high level of the available database objects. For this example, name your dataset **EmployeeDirectory** and select a database view to retrieve the data. To use the database view, you must expand the views by clicking on the plus sign and then select the `vEmployee(HumanResources)` view. Then click Finish. Figure 11.7 shows the design environment after you've created the dataset for your report. Notice that `EmployeeDirectory` is created as an XSD file in your project, which allows you to reuse this dataset in other reports, forms, or code if needed.

Step 5 Because reports are created in the Business Intelligence Development Studio environment, you should be comfortable creating your report from this point on. For this report we created a simple table report with six columns:

- Name
- Job Title
- Phone
- Email
- City
- State

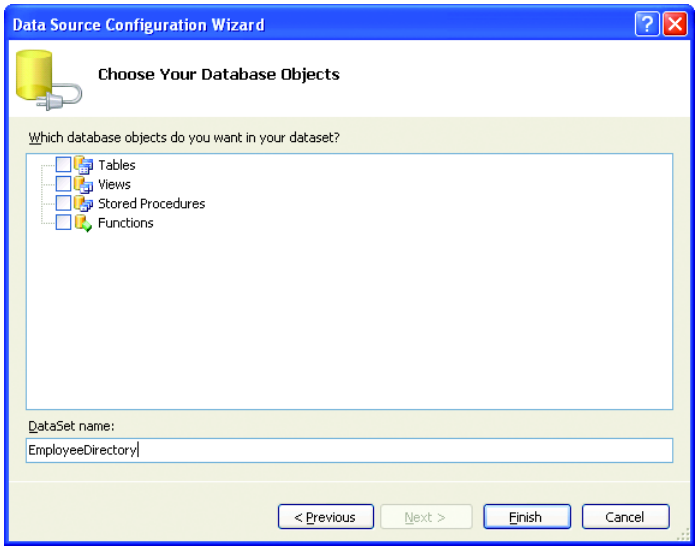

Figure 11.6
You can add tables, views, stored procedures, and functions to your dataset for use in local reports.

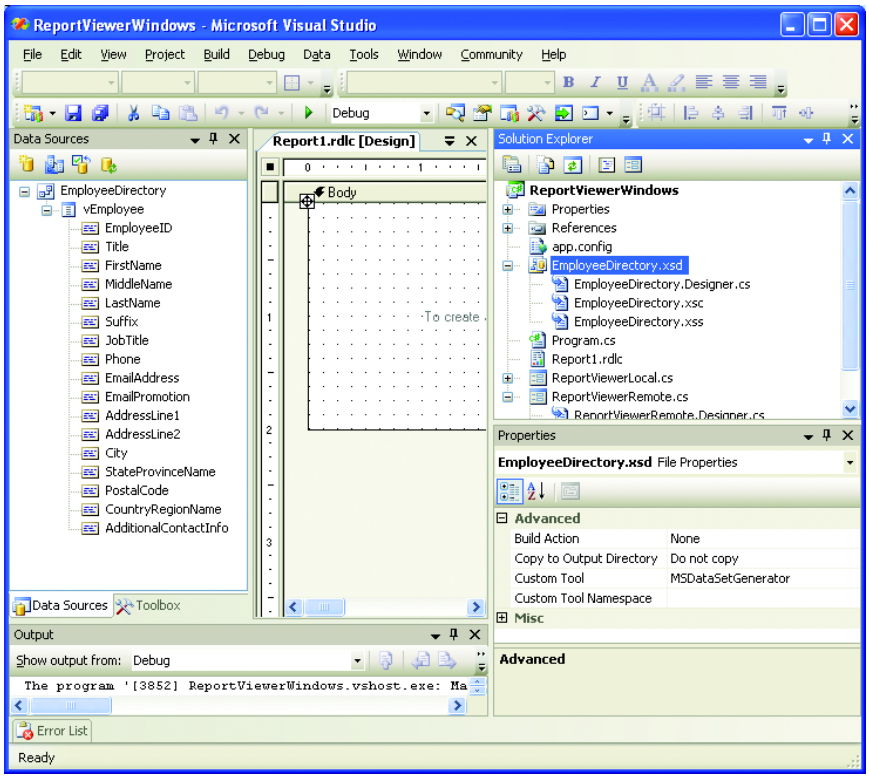

Figure 11.7 The dataset EmployeeDirectory shows up in the Data Sources section as well as in your project code as an XSD file.

NOTE For more information on creating tabular reports, see part 1 of this book.

One problem with creating local reports is that you can't preview the report in the Visual Studio IDE as you could if you were designing in a Reporting Services project. To preview your report you will need to complete step 5. For now, apply any formatting and save this report.

Step 6 Now we'll add the report to the form. First, open the `ReportViewerLocal` Windows Form, and in the smart tag window, select the report that you created to add it to the form. Figure 11.8 shows the ReportViewer task's smart tag window after you've created a local report in your project.

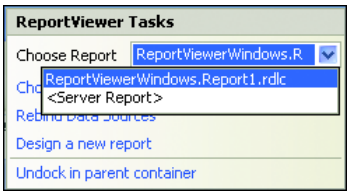

Figure 11.8 The ReportViewer lets you choose the local reports.

You have now created your first local report, which you can run without using a Report Server. If you doubt that this report is *completely* local, try stopping the SQL Server Reporting Services service and rerunning the report.

11.3.2 Creating a local report with an object as the data source

You are not limited to getting data directly from a database. Using local mode, you can get data for your reports from your .NET business objects or data objects. In fact, you may not have to do any special coding to your business objects in order to use them. Let's take a look. The chapter 11 folder in the AWReporterWin project contains a sample of a business object that gets all of the products and aggregates the sales by category and subcategory. First we will go through the objects that have been created for this example.

Business object code

The `Product.cs` file has two classes: `Product` and `ProductInformation`. Listing 11.1 shows the code for the `Product` class.

Listing 11.1 AWC.Reporter.Win product object

```
namespace AWC.Reporter.Win
{
    public class Product
    {

        private string productName;
        public string ProductName
        {
            get{return productName;}
            set { this.productName = value; }
        }
```

```
        private string productCategory;
        public string ProductCategory
        {
            get { return productCategory; }
            set { this.productCategory = value; }
        }
        private string productSubCategory;
        public string ProductSubCategory
        {
            get { return productSubCategory; }
            set { this.productSubCategory = value; }
        }

        private decimal productSales;
        public decimal ProductSales
        {
            get { return productSales; }
            set { this.productSales = value; }
        }
    }
```

The `Product` class contains four properties: `ProductName`, `CategoryName`, `SubCategoryName`, and `ProductSales`. You'll use this object when you set up your dataset for your report. Listing 11.2 shows the code for the `ProductInformation` class.

Listing 11.2 The ProductInformation object, which uses .NET generics to return a list of product objects

```
public class ProductInformation
 {
     public static List<Product> GetProducts()
     {
         SqlDataReader rdr = null;
         SqlCommand cmd = null;
         SqlConnection conn = new SqlConnection
            (global::AWC.Reporter.Win.Properties.Settings.
             Default.AdventureWorksConnectionString);
         List<Product> ProductList = new List<Product>();
         try
         {
             conn.Open();
             cmd = new SqlCommand("spGetProductSalesByCategory",conn);
             cmd.CommandType = CommandType.StoredProcedure;
             rdr = cmd.ExecuteReader();
             Product prod = null;
             while (rdr.Read())
             {
                 prod = new Product();
                 prod.ProductName = rdr.GetString
                 (rdr.GetOrdinal("ProductName"));
```

1 Returns generic list of Product objects

2 Instantiates a new Generic object

3 Puts returned dataset into SqlDataReader

```
                prod.ProductCategory = rdr.GetString
                (rdr.GetOrdinal("ProductCategory"));
                prod.ProductSubCategory = rdr.GetString
                (rdr.GetOrdinal("ProductSubCategory"));
                prod.ProductSales = rdr.GetDecimal
                (rdr.GetOrdinal("Sales"));
                ProductList.Add(prod);          <-- ❹ Adds product objects to
            }                                       generic ProductList object
        }
        catch( SqlException ex)
        {
            throw ex;
        }
        finally
        {
            conn.Close();
        }
        return ProductList;          <-- ❺ Returns list of product
    }                                    objects to caller
}
```

The code here is pretty straightforward. The `ProductImport` object has a static class called `GetProducts()` ❶. You've created this as a static method so that you can simply call this method without having to instantiate the object first. Note that this method returns a new type of object that is available in the 2.0 version of the .NET Framework. This `<List>Product` states that you will return a list of objects, but not just any type of object. The only type of object that you'll be able to put in this list is a `Product` object. This is not a requirement for your objects to work with Reporting Services, but it does provide a level of safety that was not available by using an `ArrayList`, for example. (For more information on .NET generics, see the "Resources" section at the end of this book.) The first thing this method does is set up the objects that you'll use to connect to the database. You also instantiate a new generic list (`ProductList`) that contains your product objects ❷. This `ProductList` object is the object that you'll return to the caller. You open a connection to the database and put the result set into a `SqlDataReader` object ❸. Once you have your data, you loop through each row of data in your `SqlDataReader`. Within this loop you set the properties of your `Product` object and then add the `Product` object to your list ❹. Once the loop is finished, you simply return your list to the caller ❺.

Adding and configuring the ReportViewer to use a business object data source

As in our previous examples, begin by creating a new Windows Form and this time name it **ReportViewerLocalObject.cs**. Next, stretch the form to an appropriate size for your report and add a ReportViewer control. From the ReportViewer task smart

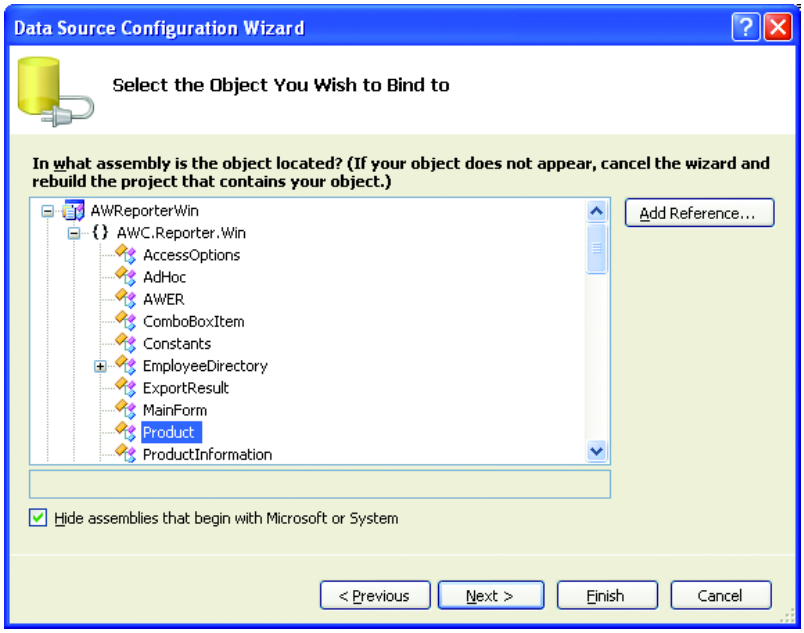

Figure 11.9 The Product object is found in the AWC.Reporter.Win namespace.

tag window, choose Design a New Report. From the Data menu select Add New Data Source to open the Data Source Configuration Wizard. Select Object as the Data Source type and click Next.

This brings up a screen that lets you select an object from assemblies on your machine as well as objects in your project. Expand the AWReporterWin project and the `AWC.Reporter.Win` namespace, as shown in figure 11.9.

Once you click Next you see a screen that displays the objects that will be added. Click Finish to complete the wizard. You should see the new `Product` object in the Data Sources Explorer, as shown in figure 11.10.

Now you're ready for the next step: creating the report.

Creating the report using business object fields

Creating a report using fields from a business object is similar to what you've done in previous chapters using database fields from a dataset. Figure 11.11 shows the report that we created by simply dragging the dataset fields onto a table entity.

As long as the properties (fields) are at the top level, you will be able to drag them just as you can with the database datasets. If you have nested objects, you have to set the path to the property by editing the expression at the field level. We show an example of this later on.

You are now ready to add your object-based report to your Windows Form. From the ReportViewer smart tag window, select the report that you just created. This not

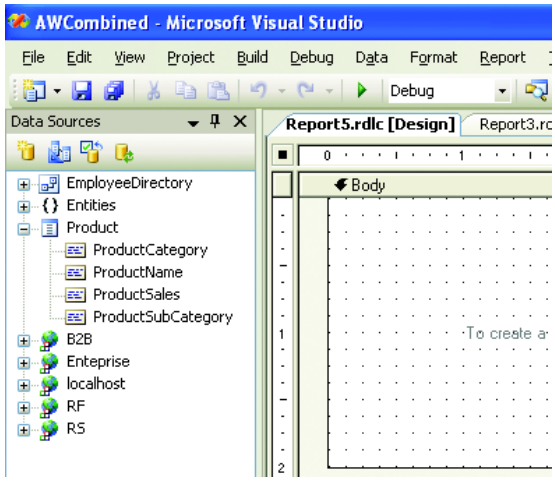

Figure 11.10
The Product object appears in the Data Sources panel.

only adds the report to the ReportViewer but also adds a `BindingSource` object to your code. In the `Load` event of your ReportViewer on the Windows Form, add one line of code:

```
this.ProductBindingSource.DataSource =
➥ ProductInformation.GetProducts();
```

This code will set the data source of your `BindingSource` object to the return value of the `GetProducts()` method of your `ProductInformation` object. Remember that `GetProducts()` was a static method—that's why you can simply call this method and you don't need to instantiate the object first. This method returns a list of `Product` objects.

NOTE The `ProductBindingSource` was created when you added the report to your ReportViewer, and since it is dynamically named, it may not be named `ProductBindingSource`.

Assuming you added the color and format to your report, after you run it your report should look similar to figure 11.12.

You've now learned how to build a simple business object and use it to create a report using the local mode of the ReportViewer control. You can now build on this example and configure a suite of business objects that makes creating reports a breeze.

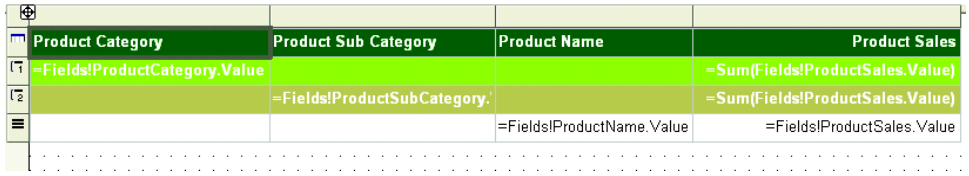

Figure 11.11 The Product Sales Report is shown here in the Visual Studio Designer.

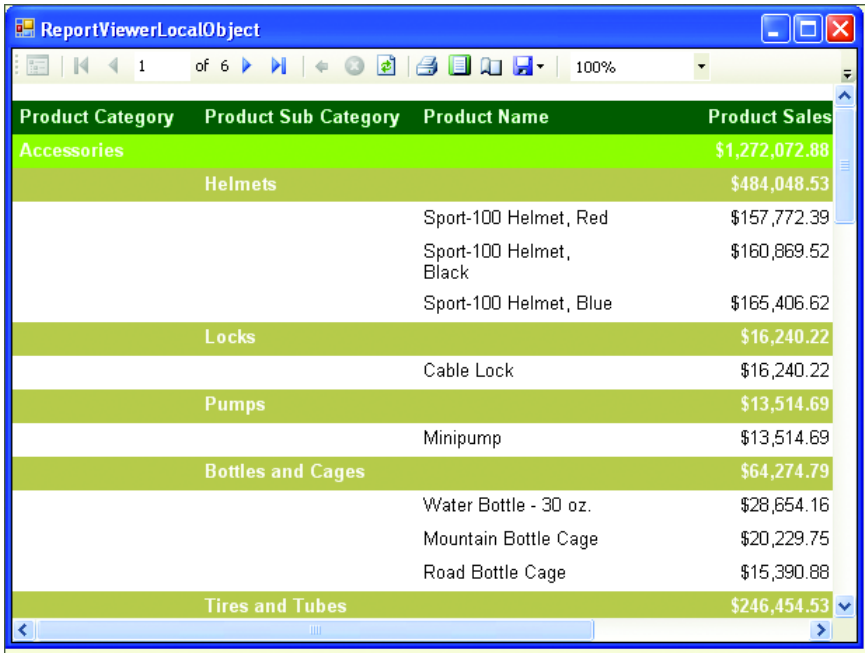

Figure 11.12 The Product Sales Report appears in the Visual Studio Designer.

Finally, we take a look at an example of implementing custom validation of report parameters with the ReportViewer controls.

11.4 CUSTOM VALIDATION WITH THE REPORTVIEWER CONTROL

The ReportViewer control adds a toolbar that provides navigation, search, export, and print functionality so that you can work with reports in a deployed application. This toolbar exposes the report parameters that you add to your reports. As you learned in chapters 3 and 10, you don't have a lot of control over parameter validation outside of simple checks. This means that by using the toolbar, you are unable to perform advanced validation of your report parameters. Let's explore one method of using custom validation of report parameters by hiding the toolbar and creating your own parameters section.

One common situation that we have run into is having to validate begin and end dates that feed the report query. Let's say that you need to make sure that the begin date is earlier than the end date for your parameters. You can implement this functionality in the following steps:

1. Create a parameters section on your WinForm and add some controls to capture date information.
2. Create event methods.
3. Write validation code.

The code for this section can be found in the `ReportViewerRemote.cs` file in the code samples provided with this book.

11.4.1 Creating a parameters section

First you must make room for your new parameters section. In section 11.2 you added a ReportViewer control to your WinForm. In this WinForm you'll move the ReportViewer control down so that you have about an inch available at the top of the form.

After you create space for the parameters section, you can start dragging labels, date-time pickers, textboxes, and buttons on the form, as shown in figure 11.3. Use the information listed in table 11.4 to add eight controls. Be sure to name the controls appropriately to match the source code.

Table 11.4 Adding controls for custom parameter validation with the ReportViewer

Name	Type	Value
`lblBeginDate`	Label	Begin Date
`lblEndDate`	Label	End Date
`lblForcasted`	Label	# of Forecasted Months
`dtBeginDate`	DateTimePicker	
`dtEndDate`	DateTimePicker	
`txtForcasted`	Textbox	
`btnRunReport`	Button	Run Report
`lblError`	Label	

When you are done, the form should look like figure 11.13. Notice that the Begin Date, End Date, and # of Forecasted Months parameters are located outside of the ReportViewer control.

Now that you have created the controls, you need to create a couple of event methods.

11.4.2 Creating event methods

The first event method you'll create is the event for when your Windows Form is invoked. This event method, `ReportViewerRemote_Load()`, shown in listing 11.3, provides you with a place to set the properties of the ReportViewer control at runtime.

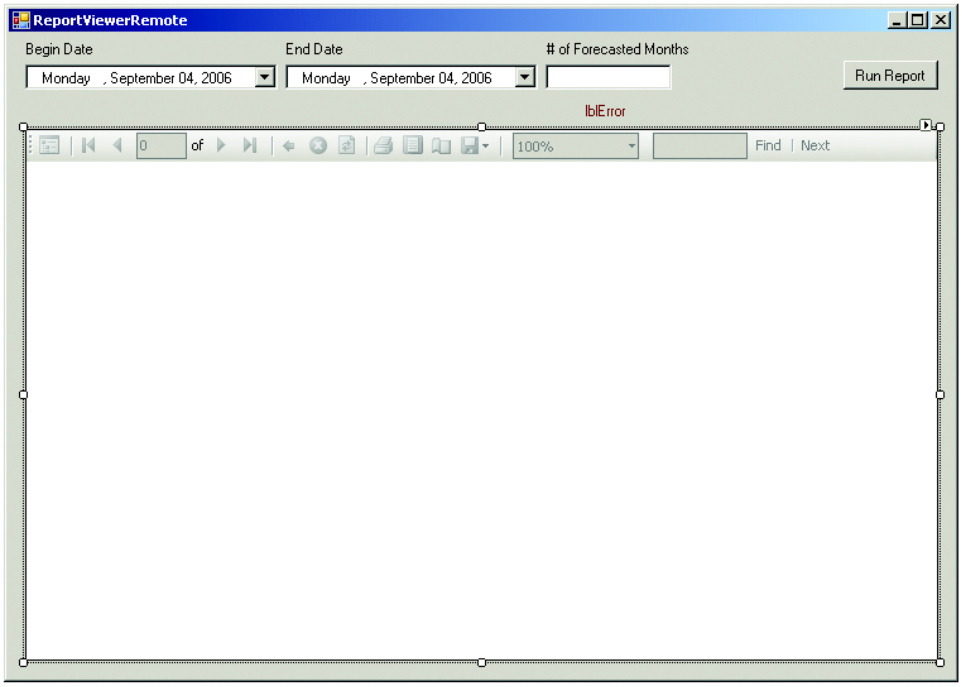

Figure 11.13 The anchor property of the ReportViewer control allows you to anchor this control to your form.

Listing 11.3 The ReportViewerRemote_Load method

```
private void ReportViewerRemote_Load(object sender, EventArgs e)
{
  lblError.Text = "";
  reportViewer1.ShowParameterPrompts = false;
  reportViewer1.Visible = false;
}
```

The `ReportViewerRemote_Load` event method does the following three things:

- It clears the error label to ensure that this label is instantiated as blank.
- It sets the `ShowParameterPrompts` property to false. This hides the `HTMLViewer` parameters section, as you wouldn't want this exposed to the user.
- It sets the visibility of the ReportViewer control to false. You want to hide the control until you've validated and collected all of the required parameters.

The second event method that you'll create will be for the action of clicking the Run Report button that you added to the form, as shown in listing 11.4. This provides you with a place to manage the validation of your parameter controls.

Listing 11.4 The button1_Click() method

```
private void button1_Click(object sender, EventArgs e)
{
  if (IsValid())
  {
    lblError.Text = "";
    //Set Parameter Values
    ReportParameter param1 = new ReportParameter("StartDate",
      dtBeginDate.Value.ToString());
    ReportParameter param2 = new ReportParameter("EndDate",
      dtEndDate.Value.ToString());
    ReportParameter param3 =
  new ReportParameter("ForecastedMonths", txtForcasted.Text.ToString());
    this.reportViewer1.ServerReport.SetParameters
  (new ReportParameter[] { param1,
      param2, param3 });
    reportViewer1.Visible = true;
    reportViewer1.RefreshReport();
   }
   else
   {
     reportViewer1.Visible = false;
   }
 }
```

Creates and populates ReportParameter objects

Adds ReportParameters to report

Refreshes ReportViewer control

Hides display of report

The method shown in listing 11.4 first checks the validity of the entered parameters by calling the `IsValid()` method. (We cover the `IsValid()` method later in this section.) If the parameters entered on your form are valid, then the code creates and populates the `ReportParameter` objects and adds these objects to your ReportViewer control. If the user entered parameters that are not valid, you set the visibility of the ReportViewer control to false.

You can easily create these event methods from the design view of your report by using one of two techniques. The first approach consists of the following three steps:

1. Go to the property tabs for the form and the button.
2. Click the Event icon in the property toolbar (indicated by arrow 1 in figure 11.14).
3. Double-click the text area of the `Load` event for the `ReportViewerRemote` form (indicated by arrow 2 in figure 11.14). Do the same for the `Click` event of the `btnRunReport` properties. This will create the event handler bindings as well as a skeleton method for your code. You can type in (or copy) your code at this point

The second technique for creating event methods is to copy the code from listings 11.3 and 11.4 into the code-behind page and then choose the pasted methods from the `Load` and `Click` drop-down lists in the property window.

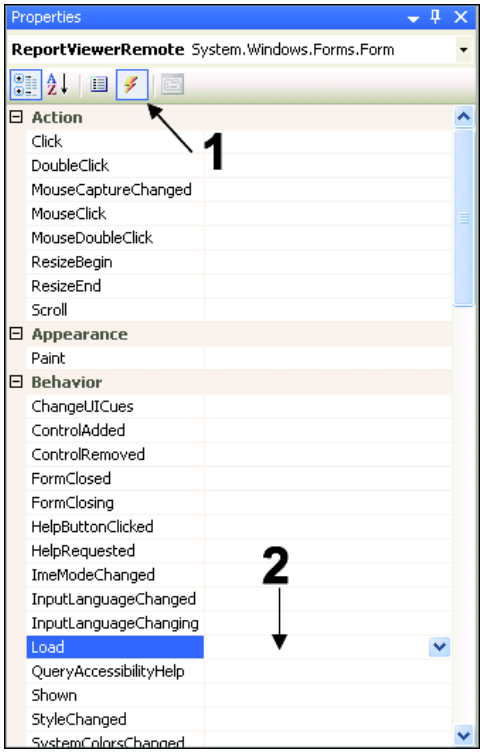

Figure 11.14
From the Events mode of the properties window, you can select existing methods for various behaviors from the drop-down list, or create an empty method by double-clicking on the drop-down itself.

In order for these methods to work, you need to add the `Microsoft.Reporting.WinForms` namespace to the `ReportViewerRemote.cs` file with the following line of code:

```
using Microsoft.Reporting.WinForms;
```

After you've created and correctly bound the event methods, the next step is to create the `IsValid()` method that will handle your custom validation for the report parameters.

11.4.3 Write validation code

As you saw earlier, the `button_1_Click()` method calls the `IsValid()` method shown in listing 11.5 to check the validity of the entered parameters.

Listing 11.5 The IsValid method

```
private bool IsValid()
{
    if (dtBeginDate.Value > dtEndDate.Value)
    {
        lblError.Text = _
"Begin Date must be earlier than the End Date.";
        return false;
```

```
    }
    if (txtForecasted.Text.Length == 0)
    {
        lblError.Text = _
 "You must enter a value for # of Forecasted Months.";
        return false;
    }
    return true;
}
```

The `IsValid()` method returns a Boolean that tells you whether the parameters are valid. If they are valid, the code in the `button1_Click` method sets the parameters by creating three `ReportParameter` objects: `param1`, `param2`, and `param3`. The `ReportParameter` object belongs to the `Microsoft.Reporting.WinForms` namespace that you added earlier. These three `ReportParameter` objects are holders for the valid parameter values and are added to the ReportViewer through the `SetParameters()` call.

You then set the `visible` property of the ReportViewer control to true and refresh the control. You must call the `RefreshReport` method to render the report.

This was a simple example of doing custom parameter validation with the ReportViewer control. We hope this will give you a jump-start on creating powerful reports with full control of parameter validation in your Windows applications.

In the next section, you learn how to use this control to add reporting to your applications when you don't have access to a Report Server.

11.5 CONVERTING REPORT FILES

As you'll recall, the local mode of the ReportViewer control uses the `.rdlc` file extension when it creates the report definition file. Reporting Services now offers you the ability to convert a local report file (`.rdlc`) to a Report Server file (`.rdl`), and vice versa. In this section, you'll see working examples in each direction. But, you may ask, when would this type of file conversion be necessary?

Let's say you created a report that was deployed to the Report Server. Now you have a Windows application that will be working with a local set of the data and will not be able to access the Report Server. This would be a good time to convert the Report Server file (`.rdl`) into a local report file (`.rdlc`).

NOTE Only SQL Server 2005 RDL files can be converted into RDLC files. If you want to convert a SQL Server 2000 RDL file, you must first upgrade it to SQL Server 2005.

Conversely, say you had some local report files that you wanted to deploy to the Report Server to take advantage of some of the features that are only available with Report Server reports, such as subscriptions or caching. In this case, you'd convert the local report file (`.rdlc`) to a Report Server file (`.rdl`).

NOTE Both the ReportViewer control and the Report Server use the same Report Definition Language schema to generate their respective report files, but the RDLC file does not contain a `<Query>` element. Even if it did, the ReportViewer would ignore it since it gets its data from the dataset defined from within the ReportViewer.

Let's explore some examples of converting these files.

11.5.1 Converting RDL files into RDLC files

For this example let's take the Sales By Territory report (`Sales By Territory.rdl`) from chapter 4 and convert it to work in the local mode of the ReportViewer. The sample project, AWConvertRDLToRDLC, is available with the source code for this book. In this section we go through the steps listed in table 11.5 to re-create this project.

Table 11.5 Steps to convert RDL files into RDLC files

Task	Description
1	Set up the RDLC file.
2	Create a new project.
3	Create a DataSet for your project.
4	Add a ReportViewer control to your form.
5	Add an RDLC file to the project.
6	Choose the report and data source for your report.
7	Configure additional properties.

As you'll see, the conversion process is pretty straightforward.

Setting up the RDLC file

The first step in converting your file is renaming it. Find the `Sales By Territory.rdl` file, copy it into a temporary location, and rename it to **Sales By Territory.rdlc**. You'll come back to this file a little later in the process.

Creating a new project

To keep the deployment simple, create a new project for this conversion. Open Visual Studio 2005, and select File > New > Project. Create a new Windows Form application and give it a name. When the project opens, you'll be presented with the default form (`Form1.cs`). For this example, let's keep this name (of course, in the real world you'd provide a more meaningful name).

Creating the dataset for your project

This step is where the "trick" comes in. In order for our RDLC file to work properly without having to modify the report code, you need to be sure that the dataset

matches the dataset that was specified for the Report Server version of the file. To do this, create a dataset and use the same SQL query that you used originally. The steps to create a dataset are as follows:

Step 1 Choose Project > Add New Item to open a screen similar to the one shown in figure 11.15. Name this new dataset **SalesByTerritory** and click Add.

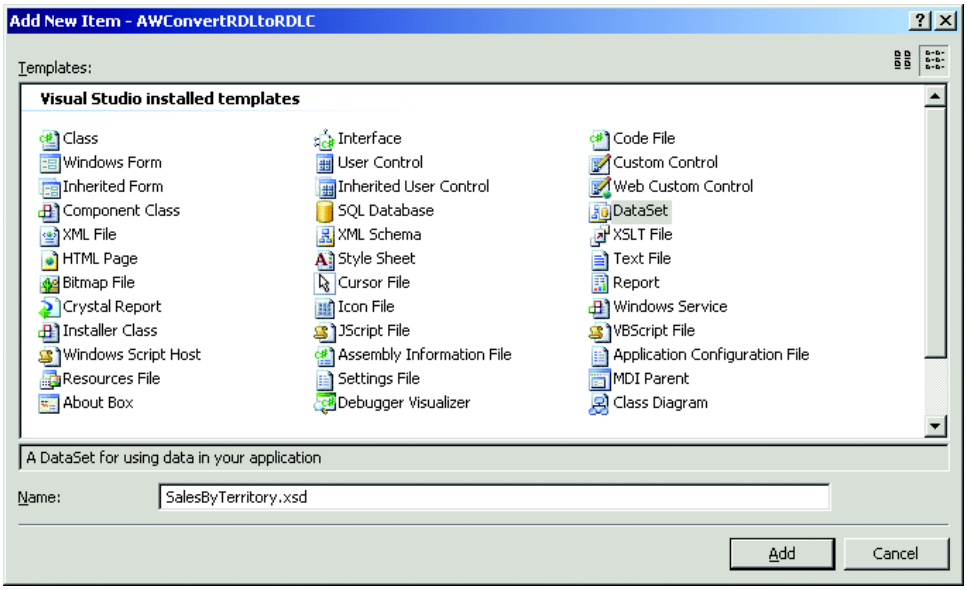

Figure 11.15 Create a dataset for a local ReportViewer report.

Step 2 With the `SalesByTerritory.xsd` file open in Visual Studio, drag a TableAdapter from the toolbox and drop it in right on the page. Doing this launches the TableAdapter Configuration Wizard.

Step 3 In the first wizard screen, set up a proper connection string and click Next.

Step 4 The second screen prompts you to name the connection. For this example keep the default.

Step 5 The next screen offers some choices for configuring how the TableAdapter will access the database. For this example, you want to use a SQL statement, so select that option and click Next.

Step 6 On the next screen (figure 11.16), enter the exact SQL statement that you used in the original report, and then click Next.

You can find this SQL statement in the example project; it's titled `SalesByTerritory.sql`. Or, you could copy this statement from the original project code. Copy and paste the SQL statement into the available space and click Next.

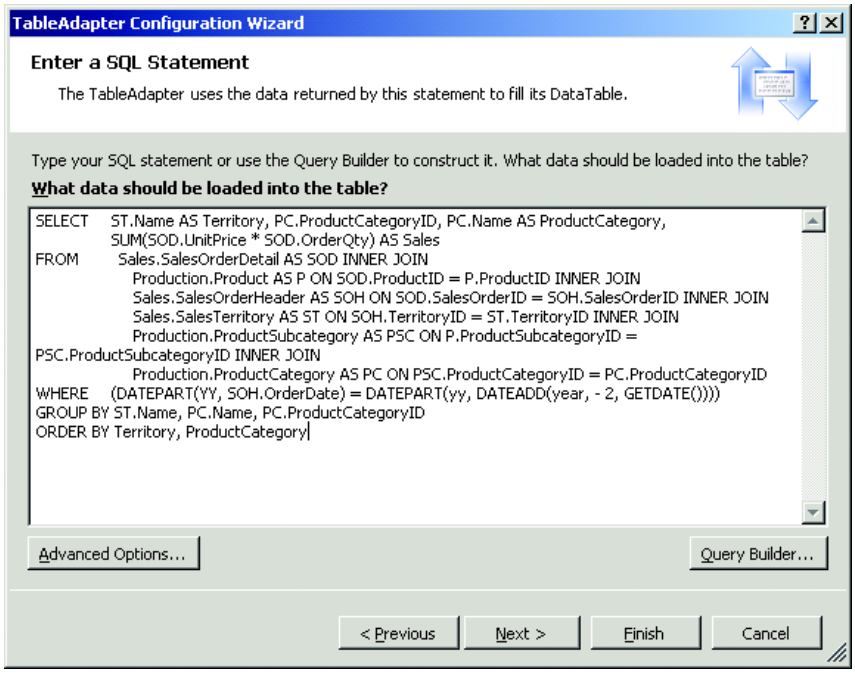

Figure 11.16 Using the same SQL statement (or stored procedure) helps keep the conversion simple.

Step 7 On the next screen, Choose Methods to Generate, keep the defaults and click Finish.

You have now finished creating your dataset. Let's move on and add the RDLC file to your project.

Adding the RDLC file to your project

From the Project menu, select Add Existing Item. Browse to the `Sales By Territory.rdlc` file that you created earlier and add it to your project.

Adding a ReportViewer control to your form

Next, open the `Form1.cs` file and stretch the form out so that there is room for your ReportViewer control. For this report, 530 pixels wide by 432 pixels high should be fine. Next, drag a ReportViewer control onto the form.

Choosing the report and data source

In the smart tag window, select the report that you added to your project and then click the *Choose Data Sources* link. Open the Data Source Instance drop-down list and you should see something similar to figure 11.17.

It is very important that you navigate all the way down to and click on DataTable1. If you choose SalesByTerritory you won't get an error, but you'll find that your report won't render. At this point for many report conversions, you would be done. However, we selected this report to work with in order to show you some additional properties that you'll need to set in certain circumstances.

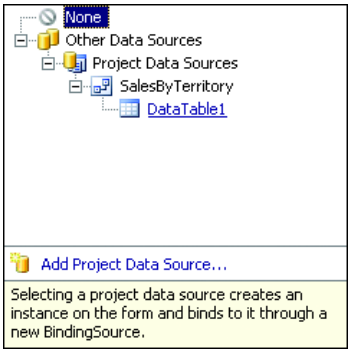

Figure 11.17 Once you have created a matching data source, you can add this to your local report

Configuring additional properties

The Sales By Territory report uses an external image for the logo. Therefore, you need to modify two properties:

- `EnableExternalImages`—Set this property to true.
- `EnforceConstraints`—Set this property of the dataset to false.

If you don't modify these properties the report won't render in the ReportViewer control. Listing 11.6 shows these properties set in the `Form1_load` method.

Listing 11.6 The Load event of the form

```
private void Form1_Load(object sender, EventArgs e)
 {
    salesByTerritory.EnforceConstraints = false;
    reportViewer1.LocalReport.EnableExternalImages = true;
    this.dataTable1TableAdapter.Fill(
    ➥ this.salesByTerritory.DataTable1);
    this.reportViewer1.RefreshReport();
 }
```

You are now ready to see the finished product. Simply run the application and the ReportViewer will show you your newly converted report. To prove that this local report is not dependent on the Report Server, try stopping the ReportServer service (ReportingServicesService.exe) and running the application.

11.5.2 Converting RDLC files into RDL files

You may find yourself wanting to convert a local client report file (`.rdlc`) into an `.rdl` file so that you can deploy it to the Report Server and take advantage of server

features such as caching, scheduling, or snapshots. Let's take the report that you created in section 11.3.1 and convert it into a format that you can deploy to the Report Server. This is a simple three-step process.

Step 1 Copy the RDLC file from the file system into a temporary directory and rename it with an `.rdl` extension. If you didn't rename the RDLC file from the example earlier, it is called `Report1.rdlc`.

Step 2 From an RS Project in Visual Studio, add the file from step 1 to the project by right-clicking on the Project menu, selecting Add Existing Item, and navigating to and adding the file you renamed in step 1.

Step 3 When the file has been imported, open it in the designer, select the Data tab, and click the Edit (…) button, as shown in figure 11.18. Update the connection information to point to the database that you want (AdventureWorks) and you are done. You can now click the Run (!) button and verify that the dataset can be retrieved.

Figure 11.18 You will need to configure the dataset connection when converting RDLC files to RDL files.

You've now converted a local report file (`.rdlc`) to a Report Server file (`.rdl`) and vice versa. There are pros and cons for using local mode or remote mode with the ReportViewer. After absorbing the content in this section, switching between modes will seem simple.

You're now ready to investigate what you need to do to successfully deploy applications that use the ReportViewer controls.

11.6 Deploying applications that use ReportViewer controls

Deployment requirements vary depending on what type of control (web or Windows) you are working with and what mode (remote or local) you use. Quite simply, you need to run the ReportViewer redistributable file (`ReportViewer.exe`) in the environment where the ReportViewer will execute. This section discusses the ReportViewer-specific requirements for deploying your web and Windows applications.

11.6.1 Redistributing the ReportViewer controls

Microsoft has provided a redistributable, self-extracting component called `ReportViewer.exe` that includes an MSI file along with other files required for a proper installation. This redistributable file can be found at `C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\BootStrapper\Packages\ReportViewer\ReportViewer.exe`.

When you run this redistributable component, the files listed in table 11.6 are copied to the Global Assembly Cache folder on the deployment computer.

Table 11.6 Files installed by the ReportViewer redistributable component

File	Description
`Microsoft.ReportViewer.WebForms`	ReportViewer control for ASP.NET pages.
`Microsoft.ReportViewer.WinForms`	ReportViewer control for Windows applications.
`Microsoft.ReportViewer.Common`	Both the Windows forms and Web server control use this for the main reporting functionality that is common in these controls.
`Microsoft.ReportViewer.ProcessingObjectModel`	This exposes the report object model to allow expressions in the report definition and access it programmatically at runtime.

11.6.2 ReportViewer deployment for Windows applications

For Windows applications, be sure to include the controls as application prerequisites so that they can be automatically installed with your application.

You choose prerequisites in the Prerequisites dialog box. To open this dialog box:

1. Select your Windows project in the Solution Explorer and then select Properties.
2. From the Properties window, select the Publish tab to open the Publish page.
3. From the Publish page, select Prerequisites.

Figure 11.19 shows the Prerequisites dialog box.

Simply select the Microsoft Visual Studio 2005 ReportViewer check box and click OK. Now when your application is installed, a check is performed by the installation

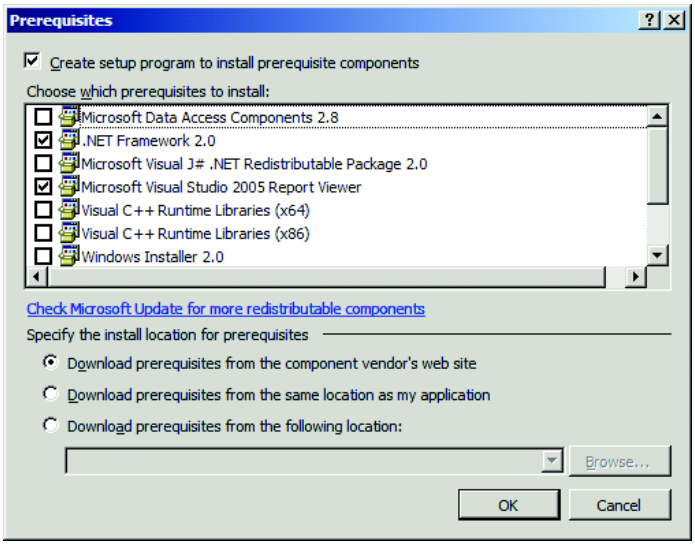

Figure 11.19 Setting the ReportViewer redistributable component will allow the bootstrapping application to automate the installation of the ReportViewer controls for Windows applications.

to see if the ReportViewer is already installed. If it is not installed, the Setup program installs it.

11.6.3 ReportViewer deployment for web applications

If you are using the ReportViewer controls in a web application, be sure that the web server has the ReportViewer controls loaded. If you have installed Visual Studio on your web server, you won't need to take any further action. Typically, though, Visual Studio is not installed on a web server unless it is a development server. In most cases, then, you have to run the redistributable file (`ReportViewer.exe`) on your web server.

11.6.4 Using the ReportViewer web server control in a web farm

You must take some additional steps if you're deploying an ASP.NET application in a web farm to ensure that view state is maintained across the farm. You'll have to modify your web application's `Web.config` file by setting the `machineKey` element. Setting the `machineKey` element forces all nodes in the web farm to use the same process identity. This is an important step to ensure that the interactive features such as drill-through will work properly.

For more information on setting the `machineKey` element, see the Microsoft .NET Framework 2.0 documentation.

11.7 SUMMARY

We covered a lot of material in this chapter and hope you have realized that your life has been made much easier with the ReportViewer control. No more adding browser controls to your Windows applications; no more adding iFrames to your web pages...

We started out with an overview of the ReportViewer control and how it works. We learned that the ReportViewer is a very handy control available with the 2.0 version of the .NET framework. The nice thing is that there is a ReportViewer control for both Windows and web applications. While there are two different controls for the two types of applications, the design UI, properties, and look and feel are virtually identical. This makes it easy for developers to switch between Windows and web applications with ease when it comes to adding Reporting Services reports. We spent a little time comparing the differences between the two controls.

We also learned that each of the controls has two processing modes: remote and local. The remote mode is used to pull reports from a Report Server and integrate them into the .NET 2.0 applications. This allows you to take advantage of the server features such as subscriptions, history, and centralized management. This works great for your applications where you have a dedicated access to a Report Server. If you do not have access to a Report Server, you can use the local mode of the ReportViewer. This means you can "unplug" yourself from the Report Server and create reports that access data from databases, business objects, and even Web services. We compared the

difference between these two modes by looking at how several features of the ReportViewer are affected by each mode. We rounded out our overview by looking at many of the properties available with the ReportViewer, and you saw when and why you might want to modify these properties.

We walked through an example of how to do custom validation with the report parameters by using the ReportViewer control. This works for both the Windows and web control and also works for both modes of these controls. Specifically, we showed you how to hide the parameters section and create your own parameter so that you can have full control of the validation and placement of your parameter controls. We modified our application to validate two date fields and ensure that the first date was chronological before the second date.

You learned in the first chapters of this book that Report Server reports that are deployed to the Report Server use the `.rdl` file extension for the report files. In this chapter we introduced a new type of report file: the local (client) report file, which uses the `.rdlc` extension. The `.rdlc` file extension is used by the local mode of the ReportViewer. You learned that both of these files hold XML defined by the same RDL schema. Since there will certainly be a need for some organizations to share reports between these formats, we found it worthwhile to spend some time showing how you can convert one format into the other.

Finally, we looked at the deployment of applications that use the ReportViewer control. You must be sure that the environment that the ReportViewer will execute in has been prepared before running the ReportViewer. You can accomplish this with a redistributable file called `ReportViewer.exe`. This is pretty simple for web applications. We covered the details of how to include the redistributable file with the application prerequisites so that they will be automatically installed with your application.

It is our hope that you learned enough from this chapter to start using the ReportViewer control in your .NET 2.0 applications. You'll see that using these controls is very simple and will save you and your organization a lot of time in integrating reports into your code.

"Helps you unlock Reporting Services."

—From the Foreword by Brian Welcker
Group Program Manager, Microsoft SQL Server Reporting Services

SQL SERVER 2005 Reporting Services IN ACTION

Bret Updegraff Foreword by Brian Welcker

Reports are the lifeline of business, so a good reporting environment is a big deal. With a powerful tool like Microsoft Reporting Services, .NET developers can add reporting to any type of application, regardless of its target platform or development language. Greatly improved for SQL Server 2005, Reporting Services now provides tighter integration with SQL Server, improved developer tools, and an expanded array of options to empower end users.

SQL Server 2005 Reporting Services in Action helps you build and manage flexible reporting solutions and develop report-enabled applications. In this clear, well-illustrated book, you'll follow a report from creation to publication. Along the way you'll explore flexible delivery options like web-based, on-demand, and subscribed reports—complete with cool new features like direct printing and client-side sorting.

For applications that require custom reporting, you'll learn to define reports with RDL and push them to the Report Server using the Report Manager Web Service API. You'll also see how to write server extensions to expand the range of data processing and report delivery options.

Written for developers with a solid foundation in .NET and SQL Server.

What's Inside

- Report Builder and ReportViewer
- Ad-hoc reporting and client-side sorting
- ASP.NET web control for server-side report generation
- Create reports off ADO.NET datasets

This book is a revised edition of *Microsoft Reporting Services in Action* by Teo Lachev, Manning, 2004.

Bret Updegraff holds the MCSD, MCAD, and MCDBA certifications, and is a 2006 Microsoft Server System MVP. Bret is the president of the Indianapolis Professional Association of SQL Server (PASS) users group.

"Insightful, professional, and full of information."
—Dan Hounshell, Telligent Systems

"Incredible technical depth, interesting examples and solutions. Very well done."
—Dave Corun
Director of Microsoft Technologies
Catalyst IT Services

"Great case studies!"
—Nuo Yan, Microsoft MVP, MCDBA

"Excellent writing—very detailed."
—Berndt Hamboeck
Team Leader, United Nations

Ask the Author


Ebook edition

www.manning.com/updegraff

ISBN 1-932394-76-1

$49.99 US/$64.99 Canada