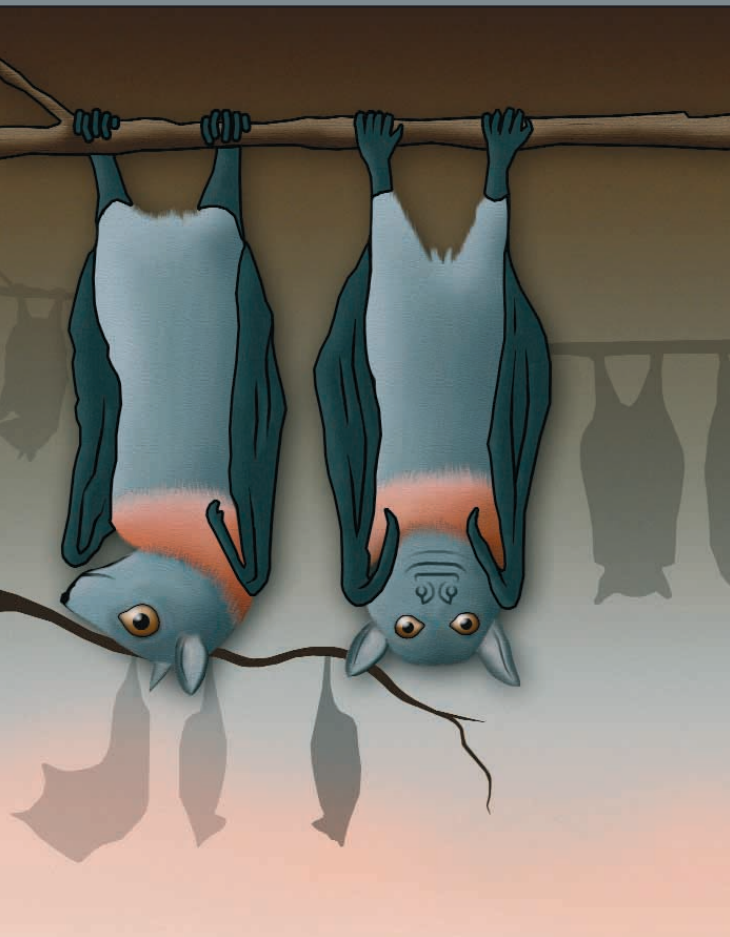


Flying Fox

*Applying Visual FoxPro Reporting
to Any Data, in Any Environment*

Lisa Slater Nicholls



dFPUG c/o ISYS GmbH

Introduction

This book gives you the tools and techniques you need to use Visual FoxPro 9.0 for reporting applications, no matter what type of database you use and no matter what type of programmer you are.

Who should read this book

Database developers who have never used Visual FoxPro can use this book to learn how to use VFP as a low-cost and full-featured reporting tool for their data sources. VFP developers can read this book to take a fresh look at reporting strategies that make use of features new in VFP 9, with a comprehensive strategy for using external data with reports.

Why VFP?

Visual FoxPro provides a great way to provide reports for your applications. Even if you don't use FoxPro for anything else, VFP makes reporting accessible, extensible, and cost-effective:

- VFP reports use any kind of data source that can be accessed through ODBC or OLE-DB.
- VFP allows you to manipulate your data efficiently, with a minimum of programming, as part of report preparation.
- When you, as a developer, ship VFP reports (FRX files) as part of your applications, you can include the customizable Report Designer, so your end-users can personalize the output.
- The VFP components you distribute to run reports in your applications, and even the Report Designer components you can optionally provide for your users, are royalty-free.

In this document, you'll learn to access MySQL data in Visual FoxPro and create reports with that data. I will provide some recommendations suitable for using external data of any type in VFP reports. I will demonstrate using some custom-built VFP tools to help you standardize common tasks for multiple reports, which are part of the source code for this tutorial.



I'll use the standard MySQL World sample database in all my examples. You'll find the full world.sql script to create this database in the source code for this document, in the "prepare_data" subfolder. You can learn more about the World sample database at <http://dev.mysql.com/doc/world-setup.html>.

If you don't use MySQL, you can follow along with the examples in Microsoft SQL Server. I have prepared a separate file, `msworld.sql`, which you can use to load the World database into your instance of MS SQL Server. Change the `CREATE DATABASE` statement at the top of the `msworld.sql` file to include appropriate paths for both the database and database log files on your system.

If you use other ODBC-compliant database servers, you should be able to edit the `msworld.sql` script slightly for their use as well.

How to read this book

This tutorial does not assume any previous VFP knowledge. If you are not currently a VFP developer, start at the first, brief chapter (*Prepare your development environment*) and read straight through to the last chapter, at whatever rate feels comfortable to you. At the beginning, you will learn to access your data interactively and easily in the Visual FoxPro development environment. In the middle, you will learn about VFP report construction and applying VFP object-oriented design techniques to reports. By the end, you will be delivering and deploying polished reporting applications.

If you are already a VFP user, you don't need help learning to use the VFP IDE, so you can begin in chapter 2 (*Bring your data into Visual FoxPro*). If you've been using external data for a long time, you may be tempted to skip chapter 2 as well, but give it a quick skim first. You may find some of the book's recommendations for data use with reports to be a little different from what you use in your current VFP applications, and chapter 2's introductory instructions will let you know what to expect.



The book's approach to data access is tuned for a reporting-centric use of external data, in cases where little or no application-specific Visual FoxPro code should be expected to surround the generation of report output. For this reason, the instructions bind the data closely to reports. They also emphasize retrieval of large, read-only data sets, as is appropriate for reporting applications.

You can substitute any VFP data-handling mechanism that you prefer after you review this section of the book.

Non-VFP developers and VFP developers who are not experienced with reports can use chapters 3 (*Create reports*) and 4 (*Customize your report layouts*) as a basic course in report design. VFP developers experienced with reports can review chapter 3 briefly its recommendations on data access techniques (sections “Add data instructions to the report” and “Improve your data environment Settings”), and check chapter 4 for any layout tips and tricks that may be new to them in VFP 9.

Chapter 5 (*Communicate complex data*) describes the use of multiple tables in reports, a sophisticated “dance” with new and subtle steps in VFP 9, thanks to the introduction of multiple detail bands for report layouts. This chapter is for everybody, since it considers various and typical external data scenarios and how best to express them in output using VFP reporting tools. Use this chapter to organize and classify your data scenarios, and to identify appropriate data presentations for each.

Chapter 6 (*Use objects to make the process repeatable*) brings VFP’s considerable OOP (object oriented programming) muscle into the reporting process. You learn to use VFP’s DataEnvironment container class and its member object types, such as cursor adapters, in ways especially suitable to VFP reporting with external data. This chapter employs some techniques new in VFP 9. Beyond that, this section and the associated class library abstracts the data-handling recommendations you’ve learned about interactively in the earlier chapters, making them available to any application.

Finally, chapter 7 (*Create a VFP reporting application*) gives you a generic object and a application framework that will run your reporting commands in an end-user environment. You customize the framework for various applications by adding DataEnvironment subclasses that are knowledgeable about your data sources. For deployment, you add the reports you’ve created plus an XML configuration file specifying these reports and other conditions (such as whether end-users have the ability to modify reports or create new ones). Non-VFP developers will see exactly how to add these classes, build a VFP executable, and create an installable package in this chapter. Experienced VFP developers may enjoy using this build-and-go approach to include external reporting add-ons packages with their existing applications or, alternatively, deploy the generic _frxcommand object and its associated XML configuration file within their standard application strategy.

Icons and special notes

I use the following Icons to help you identify some types of special comments and notes in this book:



This icon means the associated note is a slight digression from the tutorial instructions in the current section of the book. It enriches your general knowledge of the current subject, but is not required to follow the tutorial to a successful conclusion.



This icon means the associated comment is of special interest to experienced VFP developers, explaining a recommendation or choice I’ve made that they may realize is only one option of many.



This icon indicates that the associated note is of special interest to new VFP users, such as MySQL database developers.

Chapter 1

Prepare your development environment

This short chapter gets you up and running in the VFP IDE, and tells you about installation requirements for using VFP in the book's tutorial exercises.

If you have not already done so, you must install Visual FoxPro 9.0 on one of the supported versions of Windows to begin this tutorial.

To follow the final section of the tutorial, you must include the InstallShield Express Visual FoxPro Limited Edition, an optional component, when you install VFP.



VFP 9.0 is required for many of the features you'll use in this process; an earlier version will not work.



If you are already a VFP 9.0 user, be aware that use of the default Report Output Application, Report Builder Application, and Report Preview Application is assumed. If you have substituted other versions of the default REPORT.APP files, some of the dialogs you see may be different and some of the customization features in the final portions of this tutorial may not apply.*

If you are new to MySQL and want to use it for this tutorial, you need to install MySQL too, of course. You can find full documentation, including the various language editions, here <http://dev.mysql.com/doc/>. For example, the German language edition index page for the on-line version is <http://dev.mysql.com/doc/mysql/de/index.html>.



During the time I wrote this book, MySQL version 5.0 was beta-tested and released. Although there are performance benefits to the new version, there are no changes relevant to the access techniques presented here. Everything described in this book should be relevant to any version of MySQL, just as they would be relevant to Oracle, SQL Server, or other data sources.

You should also install a MySQL ODBC driver on your computer. You do not need to create any DSNs in Windows' DataSource (ODBC) Administrator tool.



The instructions in this document have been tested with the MyODBC driver version 3.51.9. They should work with MyODBC drivers versions 3.51.11 or higher, including 4.x driver releases. However, they do not rely on any 4.x driver features, such as multi-statement queries. Do not use MyODBC driver version 3.51.10, which has critical failures in VFP.

Ensure that the World database is available from your chosen database server, and you're ready to access it in VFP.

Chapter 2

Bring your data into Visual FoxPro

You're ready to get to work. This chapter shows you how to access and explore your data interactively in the VFP IDE.

When you start up VFP 9.0 for the first time, you may see a helper application such as the Task Pane Manager, as shown on the left in Figure 1.

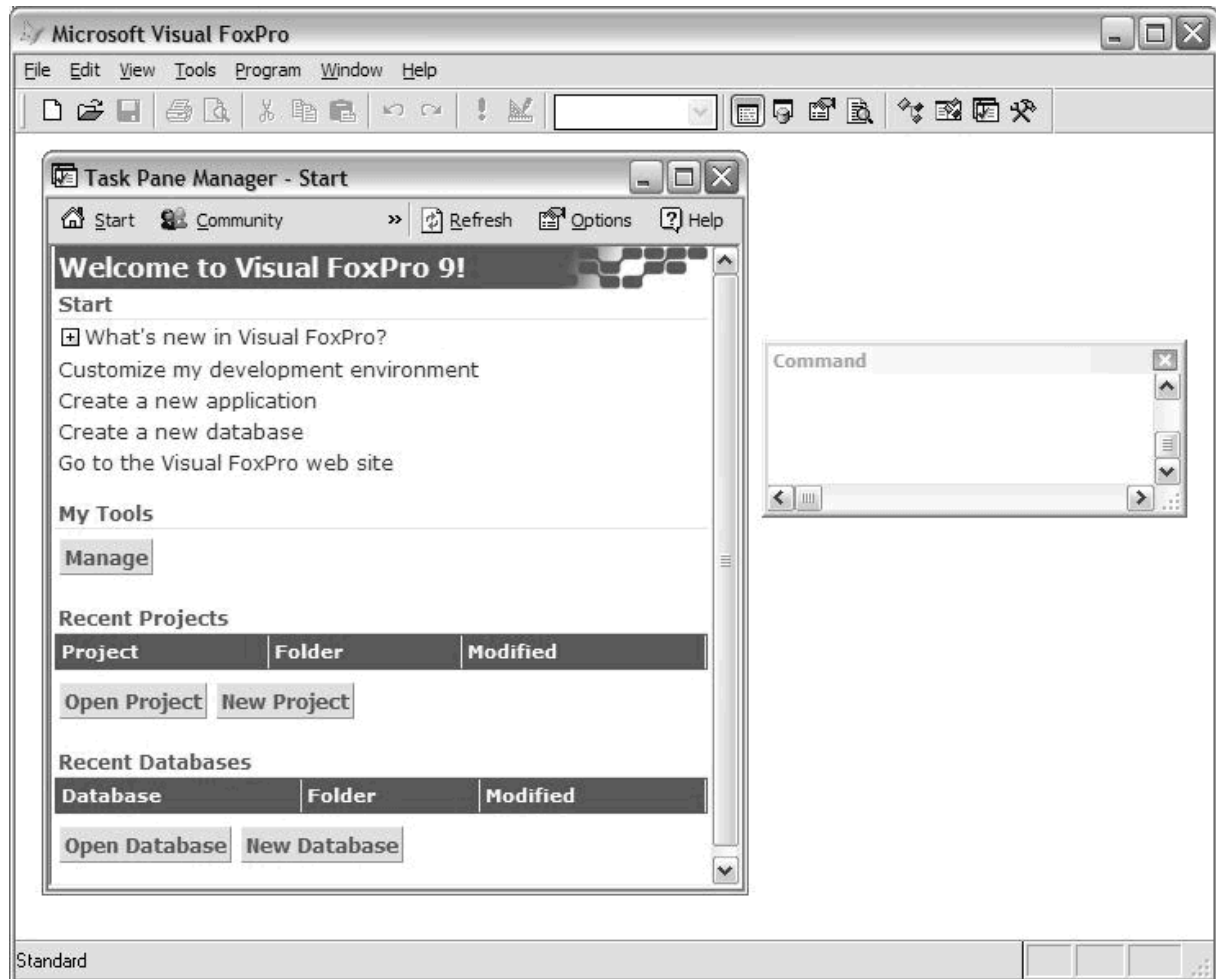


Figure 1. Starting up Visual FoxPro 9.0.

You may want to take time learning to customize your development environment and explore various ways the Task Pane Manager can help you organize your work in VFP later. For now, close the Task Pane window or any other Wizard-like interface that appears, and focus your attention on the Command Window, at the right in Figure 1. You will type a few simple commands in the Command Window to experiment with your MySQL data.

After closing the Task Pane Manager, size the Command Window so you have plenty of room. Type the command below into the Command Window, substituting your server name for **localhost** and using the correct version of the ODBC driver for your system (whether for SQL Server or MySQL).

Press Enter after the command, as if you were executing a command at the DOS command prompt.

Although it may wrap in the example below, be sure to type it all on one line before pressing Enter:

```
myString = "DRIVER=MySQL ODBC 3.51  
Driver;SERVER=localhost;DATABASE=world;"
```

Although nothing appears to have happened, this command created a variable and assigned it a value.

You can add to the value of the string stored in the variable as follows; substitute your user name and password for **XX** and **YY** as required. As before, this should be typed on only one line although it may wrap as you see it in this text (remember to press Enter only at the *end* of this command):

```
myString = myString +  
"USER=XX;PASSWORD=YY;OPTIONS=3;"
```

If you are using SQL Server, omit the **OPTIONS=3;** section of this string.

You now have a complete connection string which you can use to give VFP a connection to your MySQL or SQL Server World database.



If you are familiar with MyODBC driver settings, you may be wondering why I specified the OPTIONS flags in my connection string. This particular combination of binary option flags (1 + 2) is recommended for Visual Basic use and it works well with VFP.

Refer to <http://dev.mysql.com/doc/mysql/en/connection-parameters.html> for the full set of option flags available. For reporting purposes you may also want to add 2097152 or other option flags that improve performance.

The driver allows you to specify the flags as an expression, allowing you to list all the values and improving clarity. For example:

```
myString = myString+"OPTIONS=1 + 2+ 2097152;"
```

You can use this connection string in combination with the VFP SQLSTRINGCONNECT function, as follows:

```
? SQLSTRINGCONNECT(myString)
```

When you execute this command, you see a number echoed to the main VFP window (often called “Screen”), as shown in Figure 2. The “?” you used in the command was a print instruction, causing the return value of the function to be sent to Screen.

The return value you see in Screen is a connection handle, which you can use to access the database to which you connected until you close the connection.

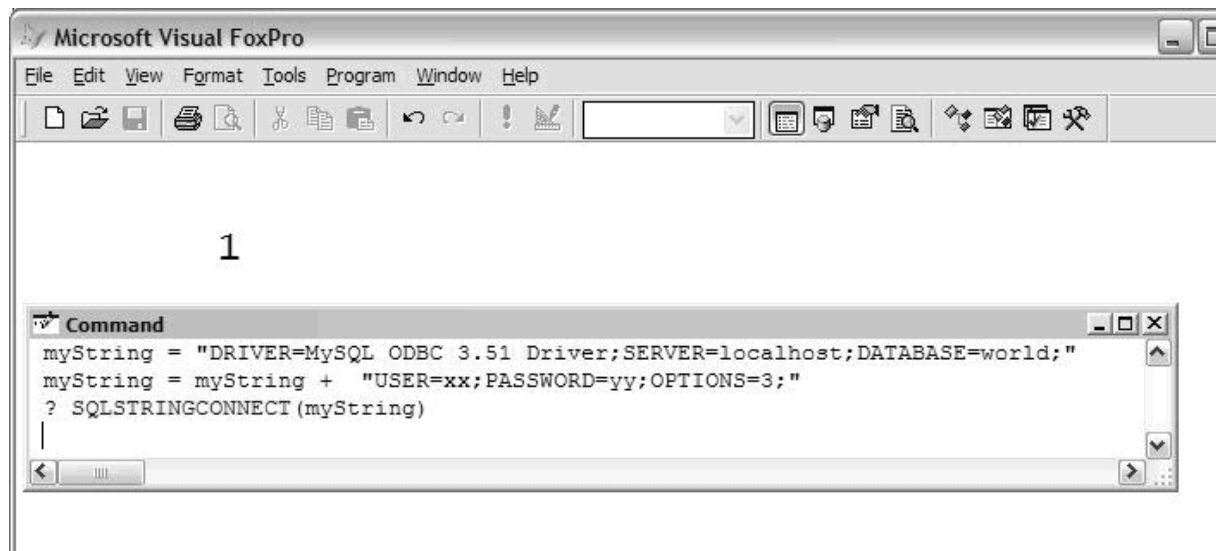


Figure 2. Connecting to MySQL.



If you are familiar with VFP, you know there are many ways besides the `SQLSTRINGCONNECT` function to set up connections to external data. You may also be surprised that this tutorial uses cursoradapters rather than cursors in many of its instructions.

I'm using the method shown here because it provides a very flexible way to design connections at runtime in an unknown environment. It does not assume setup on the client, other than availability of the ODBC driver, and does not assume that the developer uses VFP DBC containers to manage connections, etc. It also requires a minimum of code.

You can use any method you prefer, but you will discover that this technique is particularly good for generic reporting applications even if you are comfortable writing VFP code.

You're now ready to try out the data connection. Substitute the number you saw on the screen for 1 in the first command below, if VFP reported a different connection handle earlier (don't forget to press Enter after each of the two commands below):

```
SQLEXEC(1,"SELECT * FROM COUNTRY ORDER BY  
Continent, Region ")
```

SET

When you issue the `SQLEXEC` command above, if you have a slow connection to a remote machine, you may see a small window in the upper right corner of Screen as the query executes. You may not see anything happening, unless you notice the VFP status bar, at the bottom of Screen, as shown in Figure 3 below.

Chapter 3

Create reports

In this section, you'll use exactly the same syntax to access data that you used earlier, this time within a FoxPro report. You will learn some simple techniques for report design.

Before you start creating reports, you may want to navigate Visual FoxPro's current working directory to a location appropriate to saving the files you will create in this tutorial. You can use the DOS commands MD and CD in the Command Window. For example:

MD C:\MyReports

CD C:\MyReports

As usual, press Enter after each command. Enclose the fully-pathed directory name in quotation marks (either single or double) if the path name contains spaces.

3.1 Adjust the VFP IDE

Before beginning, make one change in the Visual FoxPro environment by using the command **SET REPORTBEHAVIOR 90** in the Command Window.

This instruction sets VFP 9 to use the newer of its two report-rendering engines. Although both engines will work well for most of the work you'll do in VFP reporting, the newer version requires slightly more space to render text. By specifying this style before designing a report, you ensure that the text layout controls you create are sized with enough space for use with either rendering engine.



*If you wish to make this setting permanent, go to the **Tools** menu and choose **Options...** from the menu list. In the **Options** dialog's **Reports** panel, shown in Figure 5, set the **Report Engine** behavior dropdown to 90 (Object-Assisted), and click the **Set As Default** button.*

*You can also set an option, **Century** (1998 vs 98), in this dialog's **Regional** panel, to ensure that any date expressions are sized with plenty of room. The command to set this in the Command Window is **SET CENTURY ON**.*

*When you review the **Regional** panel, you may want to ensure that Visual FoxPro defaults for date expressions and other locale-sensitive output are formatted appropriately for your language and culture. Although these items can be re-set within*

reports, it is a good idea to verify their default settings in this dialog so you don't have to adjust them individually for each expression.

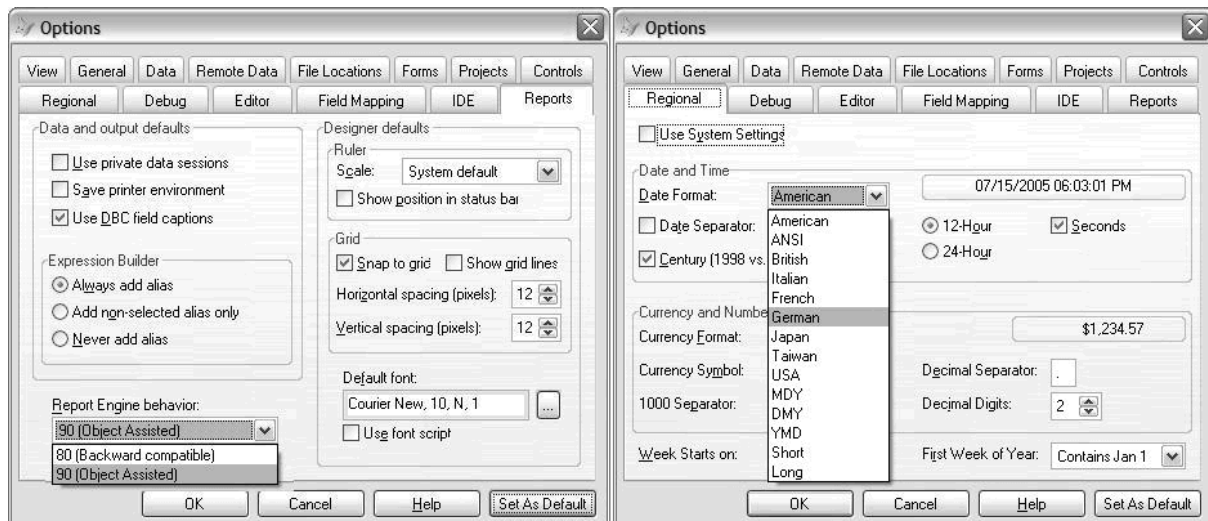


Figure 5. VFP's Tools Options dialog has a panel to set Report-specific items such as Report Engine behavior, and a separate panel with Regional settings that are also significant to reports.

3.2 Open the VFP Report Designer

To start designing the report, open the VFP Report Designer. You can do this by selecting **New...** on the **File** menu, and then selecting **Report** as your file type from the dialog. If you would like to use the Command Window, you can type **CREATE REPORT**.

When the Report Designer window opens, it looks something like Figure 6 below. You notice three “bands”, or report design areas, separated by “band separator bars”: a page header, a page footer, and a detail band. The report content you place in the detail band repeats once for every record in your cursor, and the report content in the page header and footer repeats once for every page.

3.3 Add data instructions to the report

Right-click any where on the Report Designer area except the band separator bars, and choose the **Data Environment...** option you see in its context menu.

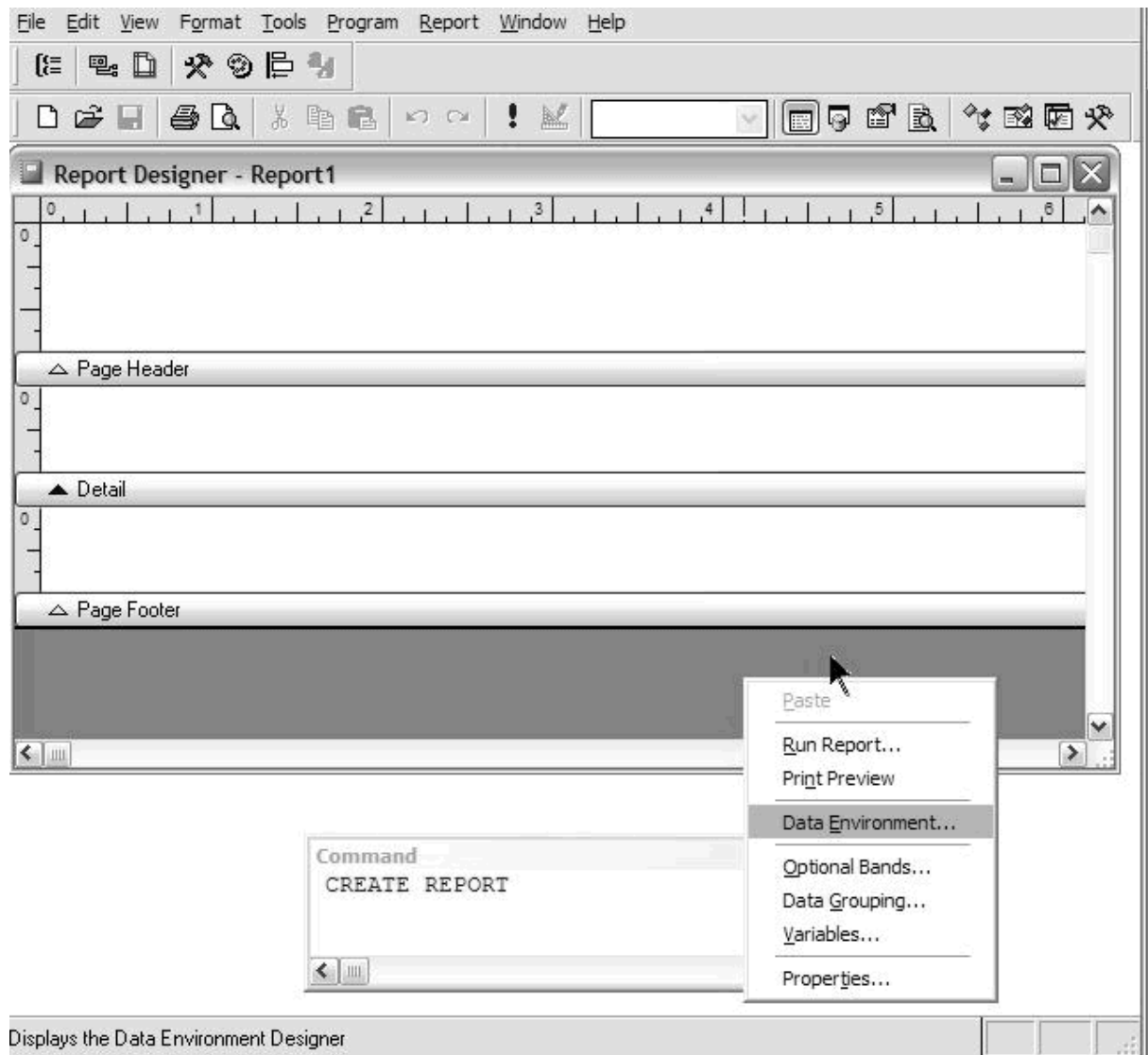



Figure 6. Begin a report design session by accessing the report's Data Environment from the Report Designer window.

After you choose this option, a new window, the Data Environment Designer, opens. It shows you another shaded design area, initially blank. When you right-click in the Data Environment Designer you get another context menu.

The first item, **Add...**, is used to add local Xbase tables (DBFs) to the Data Environment. Choose the second option on the context menu to Add CursorAdapter, as indicated in Figure 7.

 *A cursoradapter object in Visual FoxPro provides a way to handle cursors representing data that has been fetched into local FoxPro cursors, usually from external data sources. When the external data is represented as cursoradapter objects, it can be more easily manipulated as objects in VFP IDE design windows and in VFP application code.*

A “placeholder” cursoradapter appears in the Data Environment Designer window. Your next task is to give this cursoradapter object some information about the data you plan to use in the report, including the same data connection information you used in the Command Window earlier.

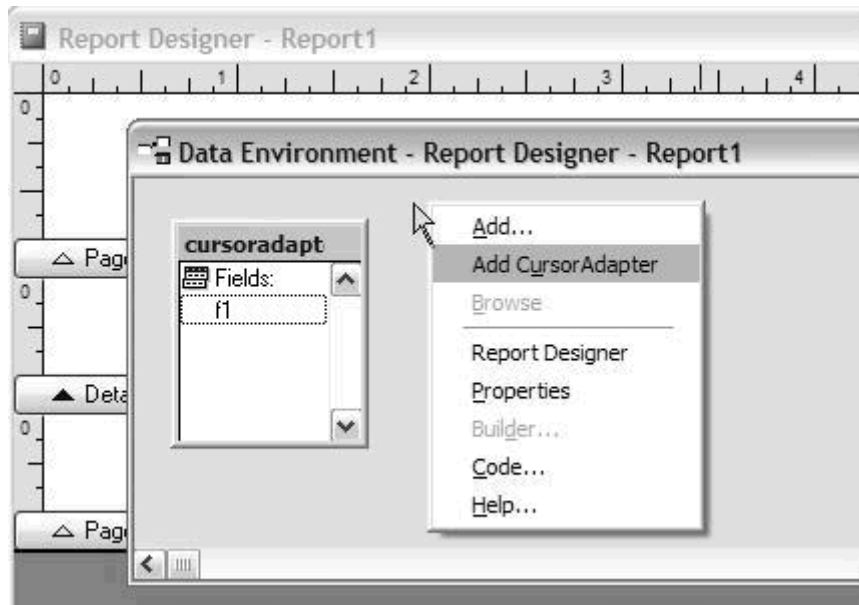


Figure 7. Get ready to add a cursoradapter to your report’s Data Environment.

Right-click again on the Data Environment Designer surface, and choose **Properties** from the context menu. You can now see the Properties Window, a tabbed palette that allows you to set the properties of the general Data Environment for your report and for each data object within the Data Environment. By selecting different objects in the Data Environment Designer with your mouse, or by using the dropdown at the top of the Properties Window, you can select which object you wish to edit. To examine general settings for the **Data Environment**, select it from the dropdown, as shown in Figure 8.

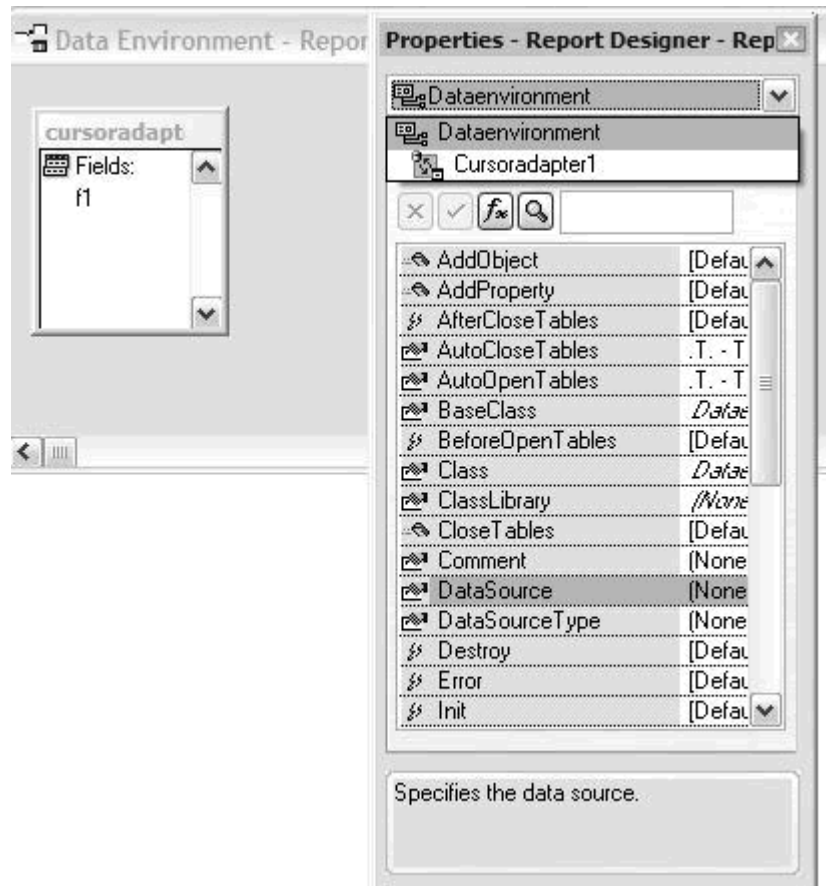


Figure 8. Edit the Data Environment in the Properties Window.

In the Properties Window, you see a list of all the attributes (on the left, in gray in Figure 8) for the currently-selected object, and the attributes' current values (on the right, in white). Above these lists, and below the object drop-down, you also see a row of small buttons and an input box.

As you highlight different attributes in the Properties Window list, these buttons and the input box change dynamically. For example, In Figure 8, they are enabled, indicating that you can type a **DataSource** value directly into the input box or press the “Zoom” button (with a magnifying glass icon) to edit this value with more space. If you highlight the **DataSourceType** value, next in the list, the button disables and the input box changes to a dropdown. This change indicates that **DataSourceType** is an enumerated type. You cycle through the available values by double-clicking directly in the value region for this attribute (which currently reads (None)) or by choosing a value from the dropdown.



*You may not be able to see exactly what shows in Figure 8 when you initially invoke the Properties Window. For example, you may not be able to see that the **DataSourceType** attribute follows **DataSource**; both attribute names may be truncated in the list.*

Chapter 4

Customize your report layouts

For your first reports, you dragged a representation of your cursor data into the layout. You can also move individual columns into the layout by dragging and dropping them from the Data Environment, so you can position them more exactly.

In this section, you learn another simple way to get your data into the layout, and then start customizing the layout more completely.

4.1 Quick-start report designs

If you have been following the instructions so far, and have not used `SQLDISCONNECT()` to remove any connections, you have at least one active connection handle to your MySQL database in the current Visual FoxPro session: the connection you created in the Command Window and additional connections created by the Report Designer when it evaluated your **DataSource** expression in the Properties Window.


For production use, you manage your connections more carefully, as you learn in a later section, but for your design sessions, it's okay to have these extra connections. You can close them explicitly with `SQLDISCONNECT()`, or they close automatically when you exit Visual FoxPro.

If you have just started a new session of Visual FoxPro, ensure that you have a connection handle available, using the technique you used earlier. Use the connection handle value in a `SQLEXEC` statement as you did before, but this time add a third parameter to the function, specifying a cursor alias for the command to use instead of the default `SQLResult`:

```
SQLEXEC(1,"SELECT * FROM COUNTRY ORDER BY  
Continent, Region ", "Country")
```

Using the Data Session Window, or the status bar, verify that you have a cursor open and selected with the alias `Country`. If it is not currently selected (highlighted in the Data Session Window's list of Aliases or displaying in the status bar), select it in the window list or use the command `SELECT Country` in the Command Window.

Now choose to create a new report as you did earlier. When the Report Designer window appears, first choose the **Default Font...** option from the Report menu, and designate your base font for this report. Next, choose the **Quick Report...** option from the Report menu. The Quick Report Dialog, shown in Figure 15 below, appears.

 In the dialog, notice the option to **Add alias**. This option, checked by default, affects the way the Quick Report generates expressions for the data in your table columns. Although for a simple report based on one cursor, you do not need to use aliases, it is a good habit to use them. You used the third argument in the last `SQLEXEC` statement to specify your alias, to ensure that Quick Report would give you the appropriate alias in each expression.

In the Quick Report dialog, you can click the **Fields...** button to get to a nested Field Picker dialog. Select some fields you used in the earlier reports. In Figure 16, I've used Continent, Region, and Name.

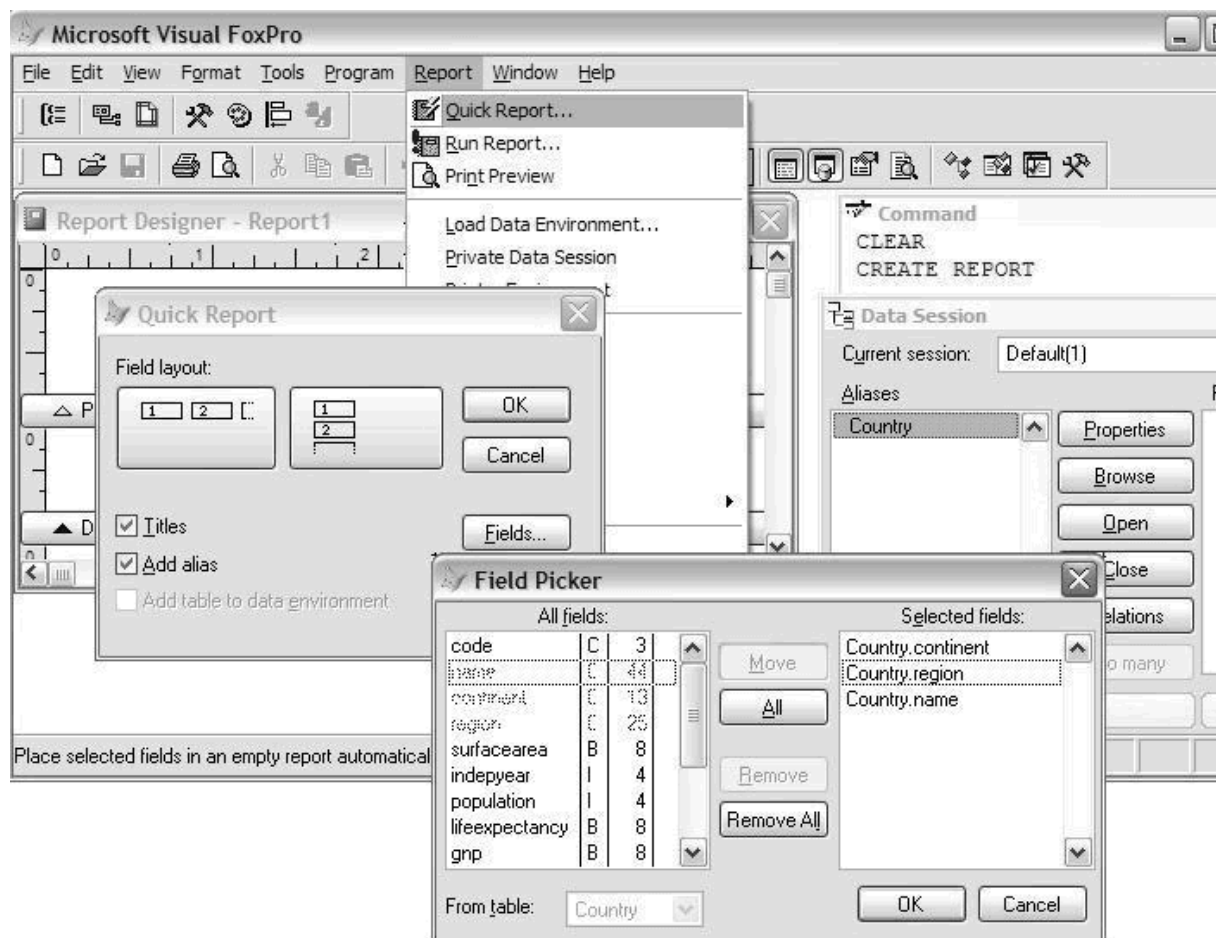


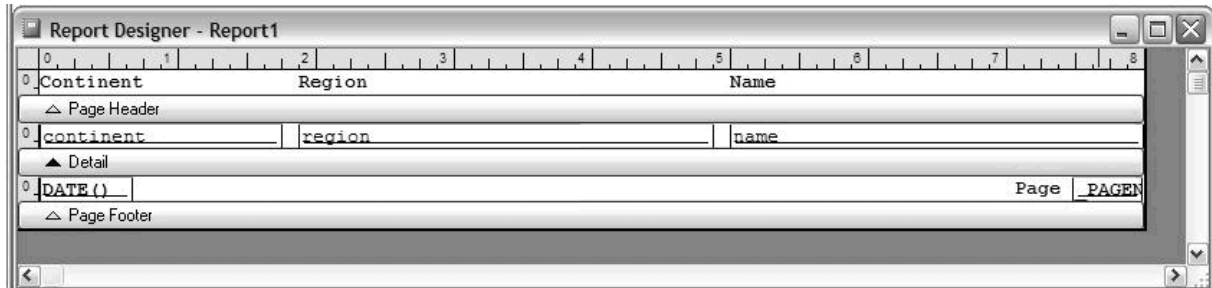
Figure 15. Building a Quick Report.

Having selected your field list, press **OK** in the Field Picker dialog, and then press **OK** again in the Quick Report dialog to generate the report contents. The Report Designer window's contents should resemble Figure 16. The exact contents depend on the default font and fields you chose.

Notice that the columns are laid out across the page, with the labels listed in bold type in the page header band, in contrast to the “form” layout you got earlier when you dragged the cursoradapter object from the Data Environment.

This columnar layout, with the column labels repeated only once per page, is appropriate for many types of reports. The Quick Report dialog gives you the option of either style of layout, by toggling the **Field layout** buttons you see in Figure 15.

Save this report as **Country2**, and preview the report. It appears as shown in Figure 17.



Figures 16 (above) and 17 (below). Quick Report layout and preview results.



As you can see, VFP finds the necessary data available, because you explicitly issued a SQLEXEC statement in the Command Window. (You created the cursor to make it available to Quick Report.)

How can VFP find the data when you run this report in an application?

In your other reports, you added the necessary instructions to the report's Data Environment. You can do the same thing for this report, but you do not have to recreate the instructions.

4.2 Load the Data Environment from a similar report

This report uses the same cursor alias and the same basic data as your previous reports. You can include the same data environment in this report as before, by choosing the **Load Data Environment...** option from the Report menu. Choose the option to **Copy from another report file** in the Report Properties dialog, and click the **Select...** button. Choose either of the reports you have already designed from the Open dialog, shown in Figure 18.

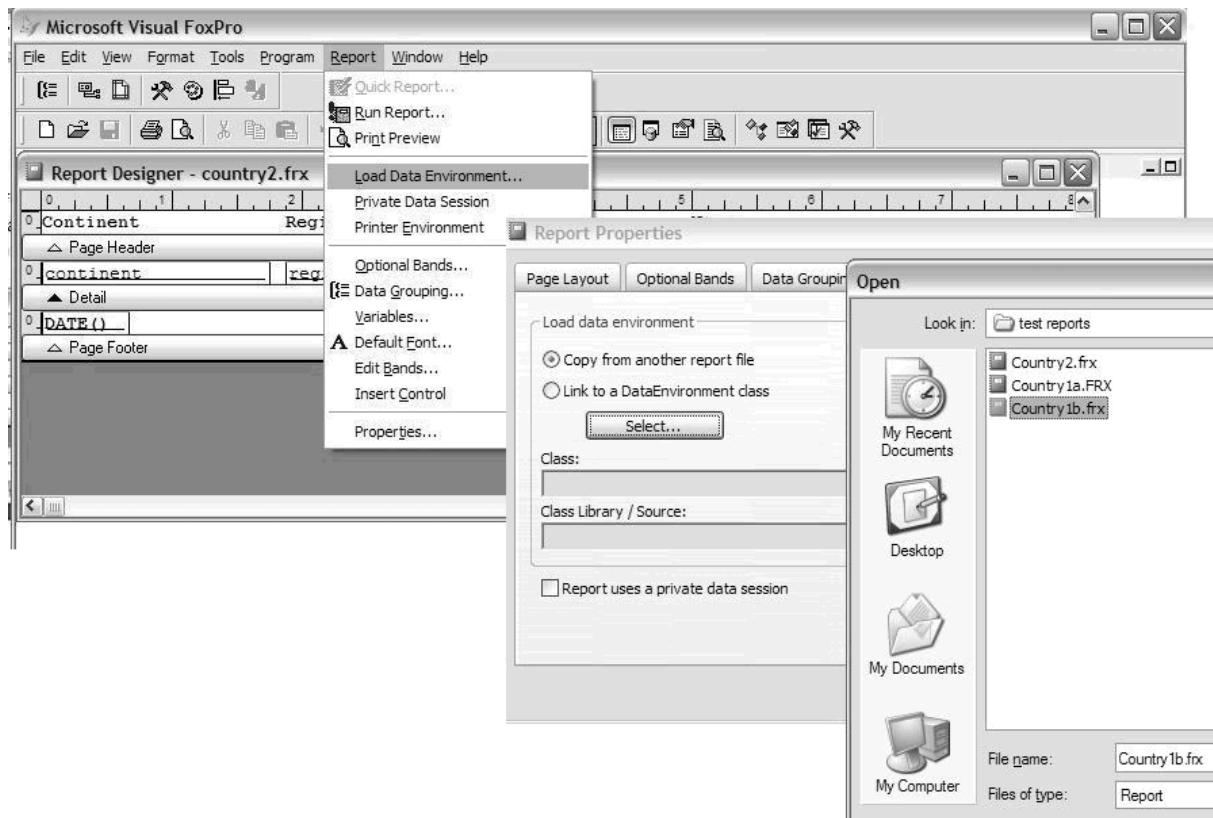


Figure 18. Loading another FRX's Data Environment.

When you press **OK** in the Open dialog you see two dialogs. The first reminds you that you are about to replace the current Data Environment in the current report (it is currently empty), and the second reminds you that you must save your changes in the Report Properties dialog to have these changes take effect. After these two reminders, you are back in the Report Properties dialog, and your chosen report filename shows in the box labeled **Class Library/Source**. Press **OK** again to save your changes in the Report Properties dialog, as you were instructed, and you are returned to the Report Designer.

The new report now contains your Data Environment instructions. Save this report again to finalize its contents.

To verify the availability of the data from the new report, first give yourself a “fresh start” by clearing out all available data and connections in the Visual FoxPro environment. Using the Data Session Window, or the command **CLOSE DATA** in the Command Window, ensure that the Country cursor is no longer open. Close the Report Designer and issue the following command in the Command Window: **SQLDISCONNECT(0)**. The “0” argument to the SQLDISCONNECT function ensures that all open connections are closed.

4.3 Test-run the report and its data-handling

This time, run the report without opening the Report Designer, to make sure the Report Designer doesn't open any connections automatically.

In the Command Window, or in Visual FoxPro programs, you run reports using the command **REPORT FORM <filename>**. This command has many options, including a **PREVIEW** keyword to use when you want to see the report on-screen. In this case, run the report from the Command Window, using the following syntax:

REPORT FORM Country2 PREVIEW



*You can replace the filename **Country2** with a “?” if you wish to use the Open dialog to find it.*

The report previews as expected. As you scroll through the contents of the preview, you notice that the Quick Report mechanism provided a page footer, with a date and page number, as well as column labels in the page header.

It might be nice to format these items a little differently, and to use them on the reports you created earlier.

4.4 Customize your report layout elements

Open **Country2.FRX** in the Report Designer. You can see the date and page number expressions in the page footer band, shown in Figure 16 above. You can double-click on the expressions to reach their Properties dialogs.

The **DATE()** function you see in the left-side expression is a built-in function in VFP. Similar to **CURDATE()** in MySQL or **GETDATE()** in SQL Server, it returns a VFP date type. (When you need a date-time value in VFP, use the function **DATETIME()**.)

You do not have to worry about converting the result of the **DATE()** function to a string for reports unless you wish to concatenate it with another string within a more complex report expression (as you learn to do in this section). The Report Engine converts the final results of all expressions, regardless of data type, appropriately for output.

You can use the Format tab in the Field properties dialog to re-format the date; for example, choose **Use long date setting**. Stretch the default width of this layout control and try your preview again, to see the result, as shown in Figure 19.

Chapter 5

Communicate complex data

We have only skimmed the surface of what you can do, both for calculations on expressions and for formatting your results.

VFP provides a huge number of built-in functions for evaluating, calculating and displaying expressions of all types. The Report Designer surface gives you extensive abilities to size, move, and format controls of the various supported layout control types: labels, expressions, lines, shapes, and pictures. The Properties dialog, which displays tabs specific to each control type, exposes these abilities.

You'll continue to see these options in use, throughout this tutorial. However, a full discussion of these language and design capabilities is beyond our scope. (The Visual FoxPro help file contains full details.)

Now that you have the basics, this section concentrates on report design changes of types that are innately tied to your source data.

5.1 Show data groups

You've seen that Print When expressions can provide visual clues in a report, to organize repeating data into groups. Not every report layout, however, can use this method to provide data grouping.

For example, the report you have been working on the last section (see Figure 23) is organized in a form-based, rather than columnar, layout. If you used Print When expressions to suppress repeated Continent and Region names, it would be difficult to see these groupings in the report results. Instead, you can create group bands, and move the expressions representing the repeating data into these bands.

Group bands “embrace” the detail band with a header and footer. They can be set to start a new page, and to re-start page numbering in a report, if you want. The layout controls in group bands can be formatted completely differently from the detail band.

You can add multiple levels of grouping, similar to the way you concatenate different expressions in the ORDER BY clause for the report data. You set up the group levels in the same order you would place the expressions in the ORDER BY clause, with the outermost group level representing the leftmost expression in the ORDER BY clause.

To begin adding groups to the report in the last section, use the **Data Grouping...** option on the Report menu. (I saved the report to a new name, Country1d.frx, in your samples, first.) Press the **Add** button in the Data Grouping tab of the Report Properties dialog, and the Expression Builder dialog appears. Type the expression that corresponds to your outermost group; in this report, it is **Country.Continent**.

After you press **OK** in the Expression Builder, your group expression appears in the listbox, and you can choose to **Add** another group expression; in this case, **Country.Region**. Refer to Figure 24 to see how the dialogs appear at this point.

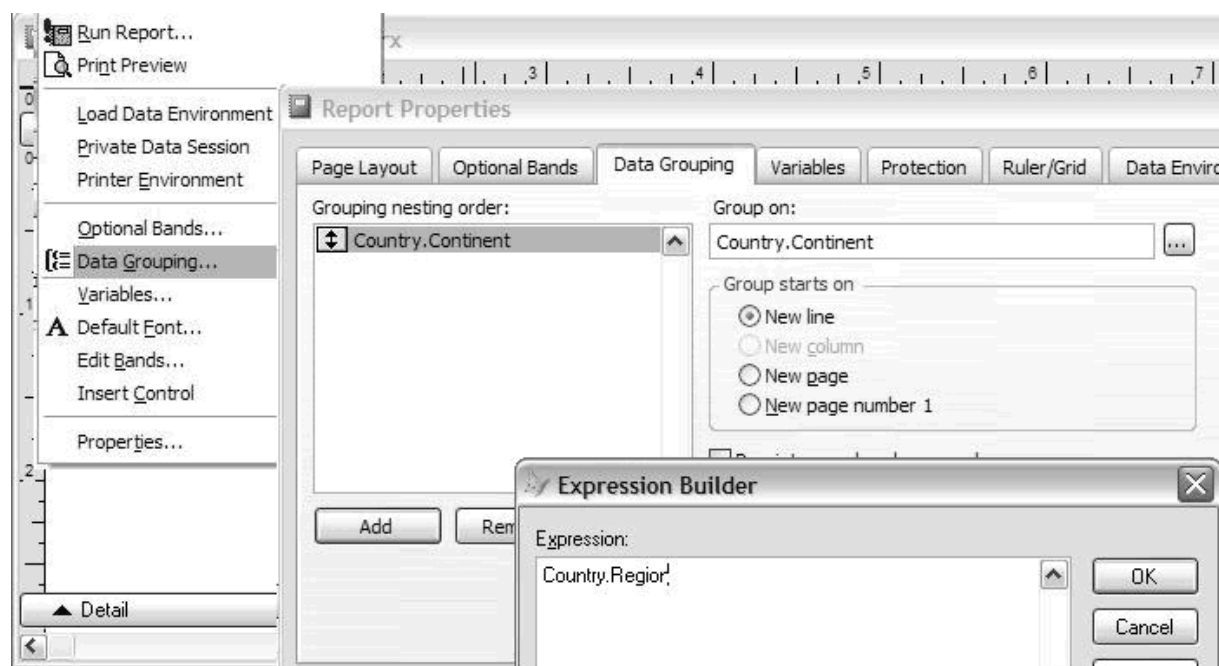


Figure 24. Adding Data Groups to a report from the Report Properties dialog.

When you have saved your changes in the Report Properties dialog and return to the Report Designer layout, it appears with two band pairs: headers and footers for your two group expressions. Drag the group header band separator bars downward, to make room for report layout controls in the header bands. The layout window should resemble Figure 25.



*If, when you return to the report layout, you find that you have specified the group levels in the wrong order, or need to make any other adjustment to the groups, you can double-click on a group header band separator bar. This action accesses the Group Header Band Properties dialog, also shown in Figure 25. The Group Header Band Properties dialog has a **Data Grouping** tab on which you can rearrange the group levels and perform the related functions available from the Report Properties dialog.*

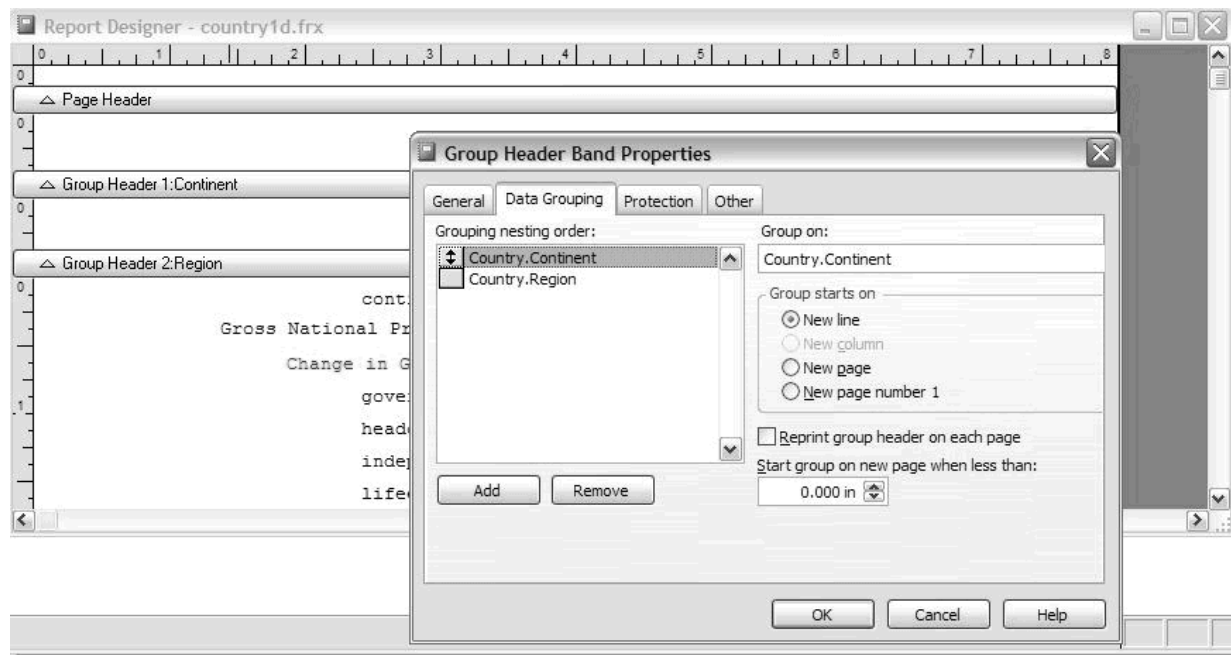


Figure 25. Group bands appear in the report layout.

Now that you have group bands available, move the associated data elements (Continent and Region into these bands).

You can add more layout elements to further indicate the group changes. In Figure 26, I've added a horizontal line to the outer group, and changed the font characteristics for group elements to distinguish them in the layout.

Now these layout controls repeat only when the group expressions change. You no longer have to use Print When to handle repeated instances in the detail band.

This report design isn't particularly pretty, but it gives you a good idea of what you can do with data groups in your layout. This ability helps you fulfill not only the requirements of repeated data in a "flat" or denormalized table, such as the World table, but also the requirements of multiple tables in a normalized data relationship.

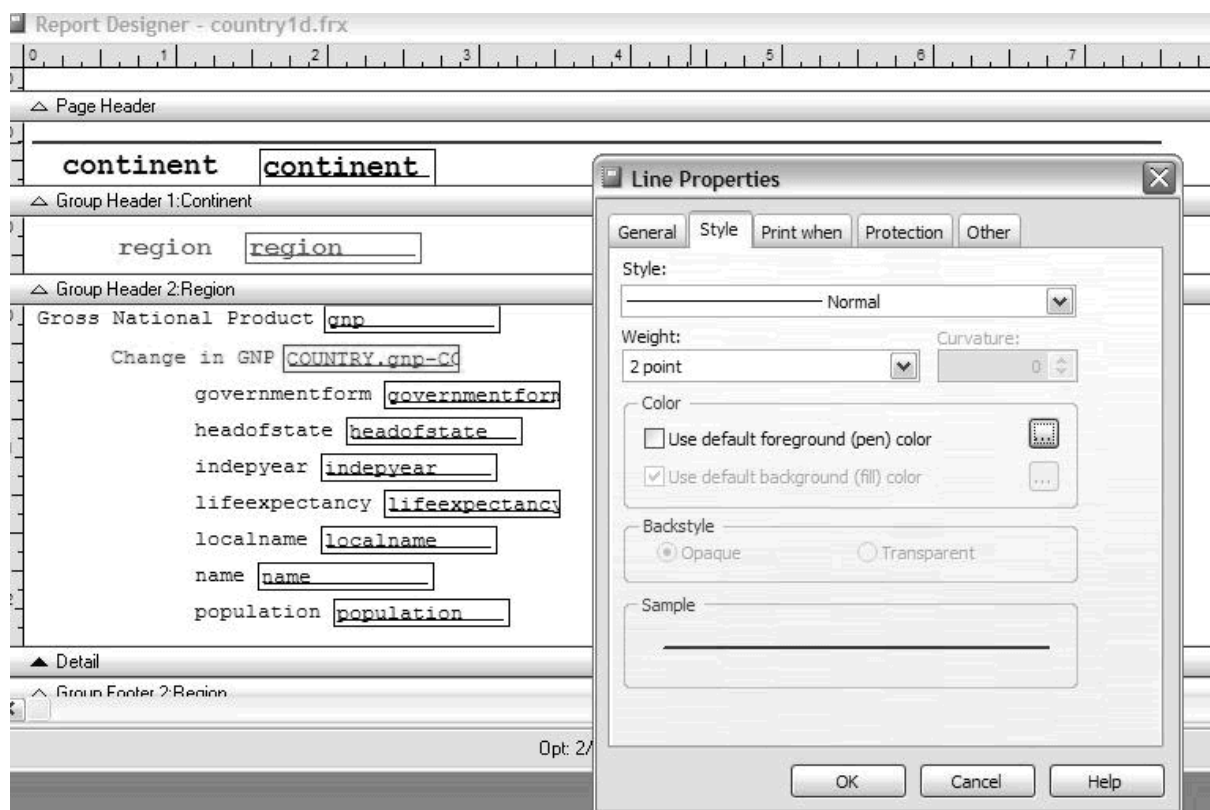


Figure 26. Add report content to group bands.

5.2 Handle multiple tables from your database

So far, all the reports you've designed handle data only from the Country table.

The World database, however, contains three tables: Country, CountryLanguage, and City. The CountryLanguage and City tables are related to the Country table through their CountryCode foreign key columns. Figure 27 shows the World database tables as they appear when examined in different applications.

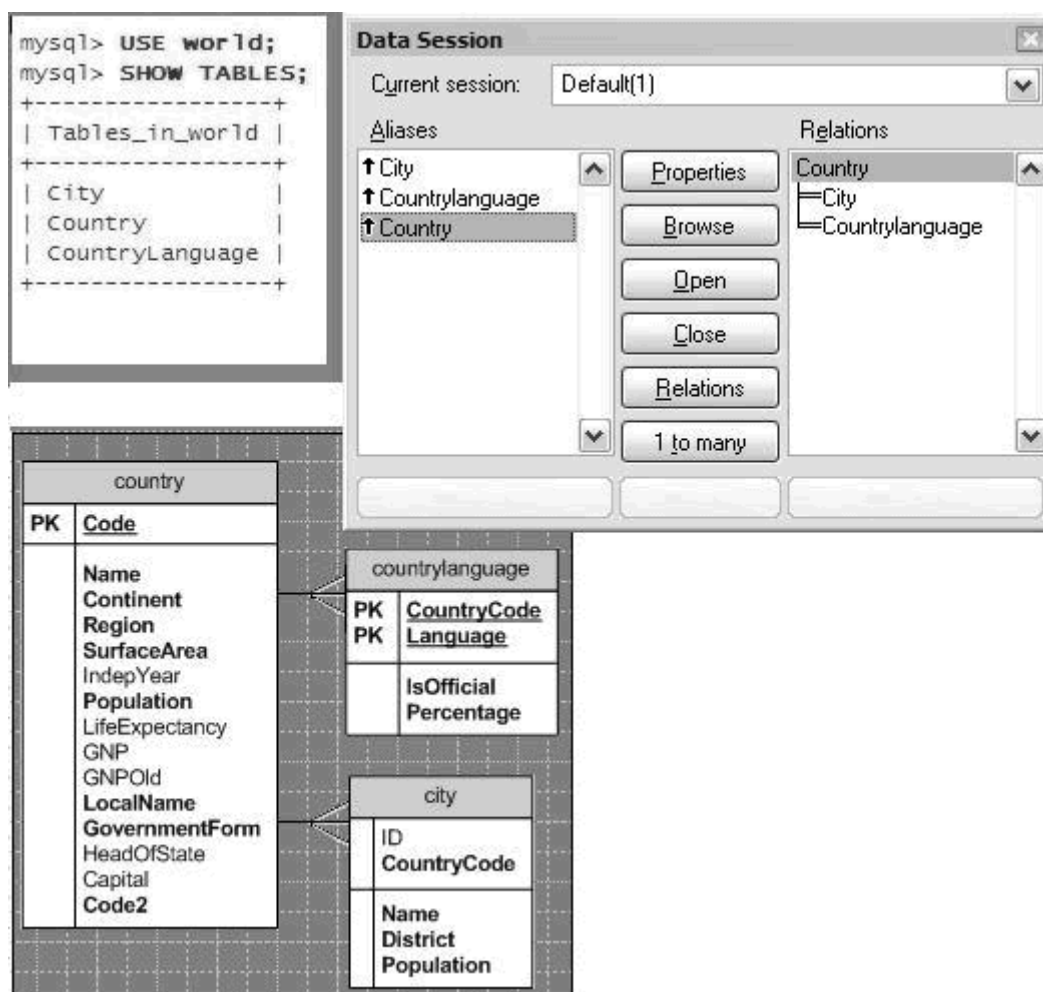


Figure 27. World data from a MySQL command line interface, in the VFP Data Session Window, and as a Visio diagram.

You have probably realized that you can create more complex reports in VFP simply by using more complex SQL statements than you've used so far in this tutorial.

Chapter 6

Use objects to make the process repeatable

The first tool you need to improve your reporting process is a `DataEnvironment` you can create outside reports, and which you can attach to multiple reports. When the report runs, an instance of this class gives consistent behavior to all the reports. In this chapter, you learn how to create and subclass an appropriate `DataEnvironment`.



Earlier, when you used the Tools Options dialog, you saw that the new VFP reporting engine is labeled “object-assisted”. However, the techniques you’re going to learn in this section work with both report engines because, in fact, reports in VFP have always been assisted by an external `DataEnvironment` object. The new engine simply allows much more flexibility and the use of many other types of VFP objects during all phases of the reporting process.

All classes in VFP are derived from a defined set of VFP base classes; the `DataEnvironment` is one VFP base class. The objects aggregated in a `DataEnvironment` container, such as `cursoradapters` and `relations`, are instances of other VFP base classes.

When you don’t specify any class to use for a report, VFP uses base classes for all the `DataEnvironment` components. By creating a subclass or derived class in a VFP class library, you have the opportunity to fine-tune various aspects of the `DataEnvironment` object used for your reports, and to put inheritance to work.



If you are a PHP programmer, you are probably familiar with the concepts of object-oriented programming; if you are a Java or .Net programmer, you certainly are!

Like these languages, VFP supports object inheritance and aggregation, although of course the syntax and some aspects of behavior are slightly different in each language.

For example, VFP works differently from PHP when you override a method in a derived class. In VFP, you can augment a method, call “up” the inheritance chain, and deliberately trigger the code in a method on various levels of the class hierarchy to control the sequence. Also, although both languages have Try-Catch structured error handling, there are some differences in the way these constructs work.

The set of classes supplied with this tutorial makes liberal use of VFP abilities and syntax. Thanks to inheritance, you won’t have to understand the behavior to put my recommendations into practice. You simply derive your classes from the ones I’ve supplied, and add your data-specific elements, such as `SELECT` commands.

6.1 Introducing your first VFP class library

A VCX class library is a separate file with the extension VCX. Like FRX files, VCX files have a companion file with the extension VCT.

To create your first VCX-based DataEnvironment class, open the sample report Country1b.FRX. As you remember, this is the report with the “improved” DataEnvironment handling you prepared earlier in the tutorial. You don’t need to save this report to another report name, because you’re not going to change the report this time; you just export the DataEnvironment information to a class library.

From the **File** menu, choose **Save As Class...**, invoking the dialog you see in Figure 46. This dialog has a number of disabled options right now (the disabled options are not used for reports). You must choose a name for your new class (I used **deCountry** in the sample) and you must also supply the name of the class library to hold the class (I used **WorldData** – the VCX extension is assumed if you don’t add it). You can also add an optional description for the class. When you press **OK** in the dialog, VFP creates the class library.



The WorldData class library delivered with your samples includes the finished versions of all the classes for this tutorial, so if you work with this library directly the contents look somewhat different from the initial screen shots here, all of which show the library being created from scratch.

While following these instructions, you can save your versions of the classes to a VCX with a different name to see results identical to the screen shots.

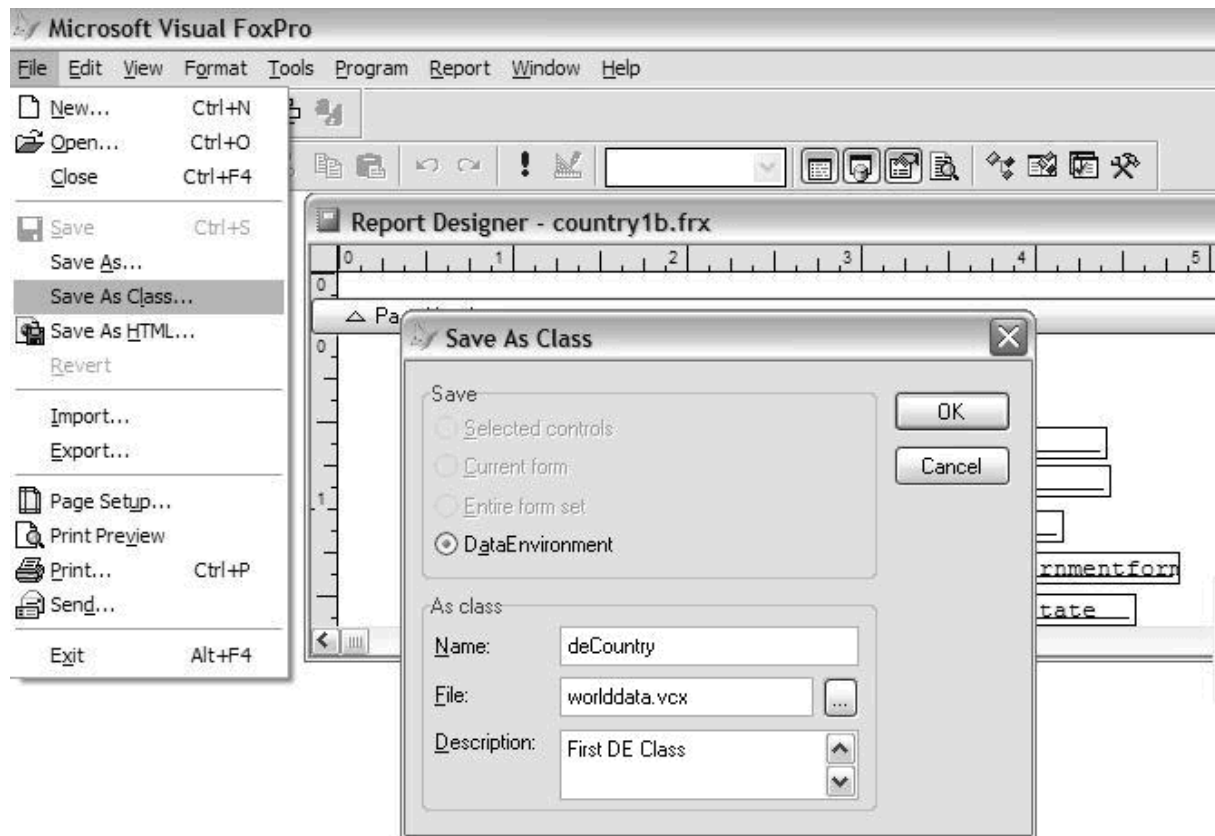


Figure 46. Exporting a DataEnvironment class definition from a report.

To see what you exported, you can open the new class in VFP's Class Designer. Although VFP offers several ways to open the class from the IDE, it is easy to find the class you created by typing the command **MODIFY CLASS ?** directly into the Command Window. A dialog allowing you to pick a class library (VCX). When you highlight the name of the class library in the left-hand list, you can then edit any class definition within the library. In your case, there is only one class in the library; the dialog looks something like Figure 47.

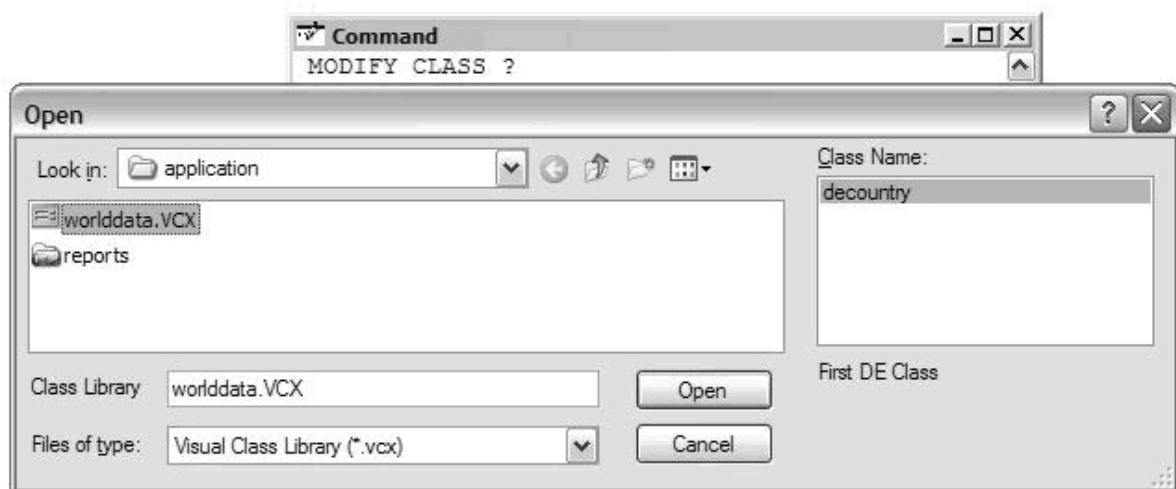


Figure 47. Opening a class in a Class library.

The new class looks significantly different from the contents of the Data Environment Designer you used earlier, as you can see in Figure 48. But, as you explore them in the Properties Window, you find that the attributes for the Data Environment parent object and its cursoradapter have been exported correctly.

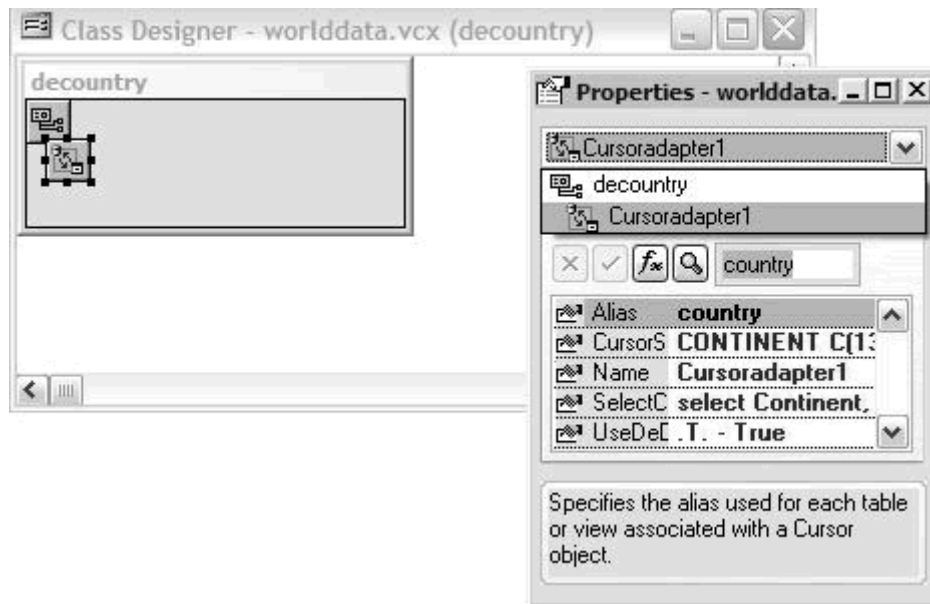


Figure 48. A Data Environment as a visual class, with a cursoradapter member.

Make a change to this class, by adding a field from the Country table that you didn't include earlier. (In the sample, I've edited the **SelectCmd** property to include the expression **UPPER(Name) AS UName** and added **UNAME C(44)** to the **CursorSchema**.) Then save the class and close the Class Designer.

6.2 Adding your VFP DataEnvironment class' data-intelligence to a report

Now you can use this class to give information to one or more reports. To test it, start with any of the Country*.FRX reports you created earlier (I used Country3.FRX). Save it to a new name (I used Country_With_Class). Use the **Report** menu's **Load Data Environment...** option again, but this time choose to **Link to a DataEnvironment class** before you press the **Select...** button in the Report Properties dialog. This time, you have access to the same dialog you used earlier to pick a class to modify in the Class Designer, rather than the standard File Open dialog, so you can pick your new class library and your new class.

You must confirm that you intend to replace the contents of the Data Environment for this report, as you did earlier. When you press **OK** in the Report

Properties dialog, and check the Data Environment Designer, you should see the difference.

You can quickly verify that the field you added to the **CursorSchema** is available in the new Data Environment, as I've done in Figure 51 below. But, when you examine the attributes of the Data Environment and its cursoradapter more carefully, you notice that it isn't exactly the same as what you saw in the Class Designer, which was a reasonable facsimile of the original Data Environment attributes in the FRX from which the class was exported.

Some of your instructions, such as the **SelectCmd**, appear to be missing, and there are quite a lot of methods with code in them with a heading similar to what you see in Figure 49.

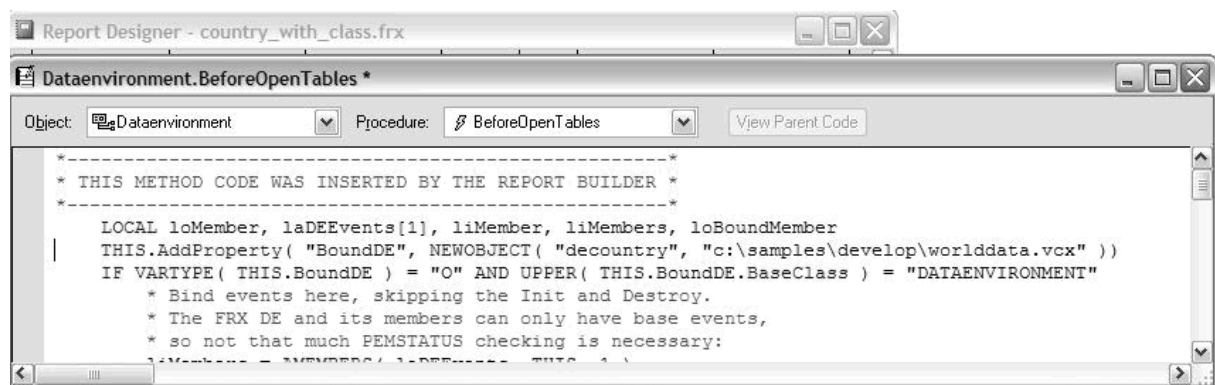


Figure 49. Report Builder-generated code in the Data Environment.

Don't be concerned. The Data Environment you see visually represented in the Data Environment Designer is now just a placeholder for the design session. When you linked the report with a class, the Report Builder (which provides the Report Properties dialog for the Report Designer) wrote some additional code to instantiate your actual class, from the class library, to provide your real instructions, when you run the report.

You also may notice that the Report Builder stores complete path information for your Data Environment class; you find this reference in the BeforeOpenTables generated code (shown in Figure 49).



If you have VFP 9 SP 1 or later, the Report Builder offers an option to omit this path information. In your sample code, I have removed the complete path information that the 9.0 original release Report Builder put in. The complete path information should not pose a problem at runtime in your applications, if you use the methods I recommend later in this tutorial. However, it may be an issue during development if you move your report around or rename a directory.

You might have to re-locate the class library for the Report Builder when you open my version of the Country_With_Class report, before you can preview this report. Although I have removed the full paths from the Report Builder code in this sample, the two files (class library and report form) are in two dif-

Chapter 7

Create a VFP reporting application

There are as many ways to create an application to deliver VFP reports as there are people who program in VFP. In this tutorial, my goal is not to teach you about all of them!

You have already learned to use VFP to create exciting and rich report layouts that leverage your data. Now you just want to get this layout in front of your users. My goal is to give you a simple set of steps that help you do it.

You will use some additional components I've supplied, along with the `_deabstract` class you have already used. You use these components to compile a generic executable (EXE) and a DLL. The EXE will allow command-line and ActiveX use of your reports, and the standalone DLL will allow some ActiveX use (without delivery of the EXE if you don't need it).

The EXE and DLL will include the VFPReports class library, your customized class library (such as WorldData), and some additional customizing instructions found in easily-edited text files.

You will deliver the EXE and/or the DLL, the supporting Reporting Application components and VFP runtime files, and your reports, to customer sites, using a standard SETUP.EXE I'll also show you how to build.

To set up your data and run reports, or to allow your customers to modify reports, you (or your external applications, such as PHP scripts on a web server) will create sets of XML instructions, describing any setup conditions and the reporting task or tasks to be performed. You will supply the EXE or ActiveX component with the XML instructions as a string or filename, and these VFP applications will run your tasks.

You can supply the XML as external files along with the VFP applications, and allow your users to run the actions simply by dragging and dropping these files on the executable, or on a script file, if you like. Because the XML is external to the applications, you will also find that it is easy to generate completely new sequences of reports, and to target different output devices, on-the-fly, according to your users' needs.

That's the plan. Let's get started.

7.1 Introducing the `_frxcommand` object

As is typical in VFP development, you need a complex, purpose-built object designed to do most of the work "behind the scenes". As you'd expect in object-oriented development, you don't need to investigate this class or understand how it works to use it.

The object in question is `_frxcommand`; you'll find it in the same `VFPREPORTS.VCX` class library as the `_deabstract` class you've been using. Its job is to understand the XML request format I'll describe shortly, and to parse the XML to run your reporting tasks.

Unlike `_deabstract`, `_frxcommand` is not an abstract class. You don't derive additional, data-knowledgeable classes from it or add code to it; you use it directly for any reports and all data scenarios.

If you open `VFPREPORTS.VCX` in the Class Browser and examine its exposed properties and methods (see Figure 72), you won't find very many.

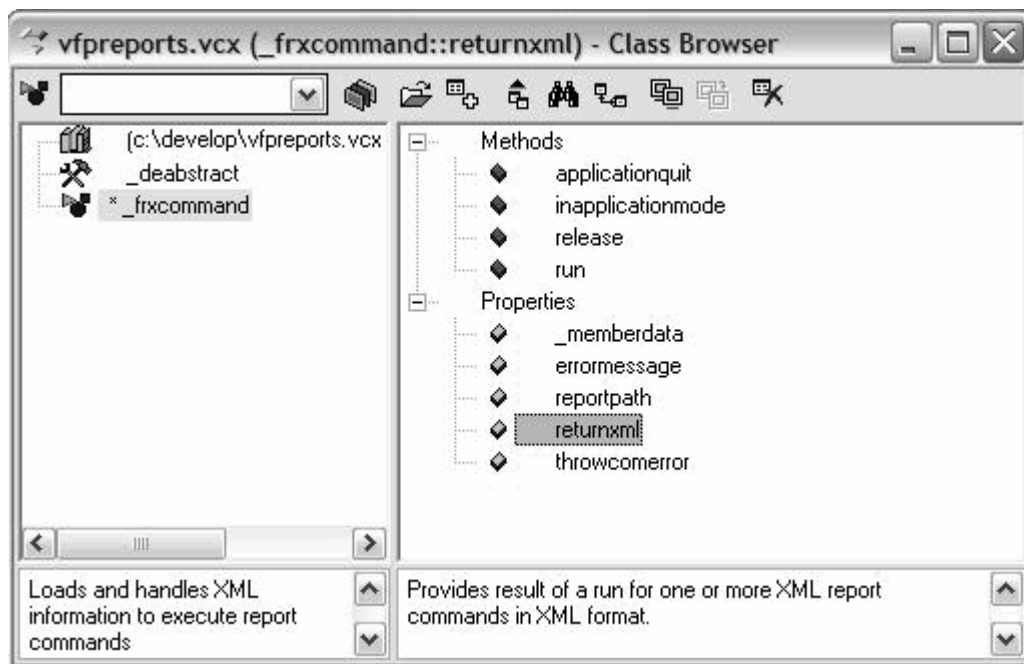


Figure 72. The `_frxcommand` object in the Class Browser shows few exposed properties and methods.

To use it, you need to know about even fewer properties and methods than you see in the figure. If you use it wrapped in an EXE file, you don't need to know about them at all. In this section, you build the EXE and ActiveX components for the first time, and use them immediately.

7.2 Building a VFP reporting solution

As delivered, your sample files include a "develop" directory with the files shown in Figure 73. Along with the generic VFPReports class library (VCX and VCT files), you see the fully-developed WorldData class library with all the data-customized classes created in earlier parts of this tutorial.



*If you use the **CD (GETDIR())** command in the Command Window to navigate to the "develop" directory, you will save some time and confusion. If you don't do it, the build process you are about to run will still ensure that your current directory is this directory before it starts to work.*

When you create new reporting solutions based on this method, with different customized classes for different data sets, and different reports to go with them, you will create each solution in a separate directory. You can use the files you see here (except the WorldData class library) as a starter set of files, each time.

When you build a VFP Reporting Application for distribution, you want it to contain your custom classes for your data set. In the sample files for this tutorial this library is WorldData. For each application with different data, you'll probably use a different customized class library.

You customize each application's build process by adjusting some constants, including the name of your custom class library, in the file shown in Figure 73 as defines.prg. As you remember from looking at the UDF_GetDetails utility earlier, PRG is the default VFP extension for a program, or command file.

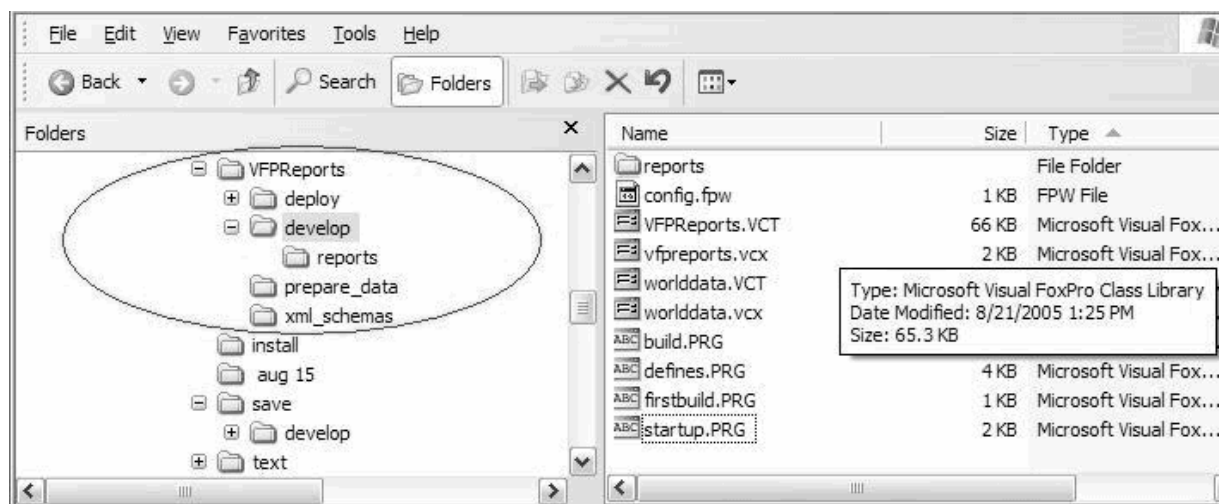


Figure 73. A Windows Explorer window shows sample folders for this tutorial, circled in red on the left, and the files initially available in the "develop" file folder, on the right.

You can open DEFINES.PRG file (remember, VFP and Windows don't care about preserving case!) for editing by choosing **Files of type Program** from the **File Open...** menu option. You can also type **MODIFY COMMAND ?** in the Command Window, as I have in Figure 74, below.

As you can see in the figure, this program file is basically a list of constants, an "Include" file for both programs and class library files.

If you are not familiar with VFP, you just need to know that VFP has a compile-time strategy for including files similar to PHP's **include <filename>** syntax and that of most other languages. The file extension does not have to be .PRG for these files, and most people prefer to use .H, for header file, as a convention.



If you are a VFP programmer and you wonder why I've used .PRG rather than .H in this instance, I've used DEFINES.PRG as the main program for the separate ActiveX-only project. I've also included EXTERNAL statements in this file, forcing external libraries to be pulled in during a compile. It also seems to force the Project Manager to "notice" any changes in the #DEFINED constants during a recompile.

DEFINES.PRG includes lots of features you can customize, such as all the text strings used by the application, which you will want to translate and personalize later. For now, just adjust the first two lines.

The first constant, MY_APP_NAME, which shows as "My Reporting Application" in the figure, is an easy way to "brand" various dialogs in the application, so change it to something that suits your needs.

The second constant, MY_APP_CLASSLIB, holds the name of your customized class library, so change it to whatever name you gave your library as you worked. You should not include the extension or the file path for this value.

When you have adjusted these two values, save this file using the same menu options or keystrokes you've used previously to save your work in a report design session. You are ready to build your application in two forms: an EXE and a DLL.

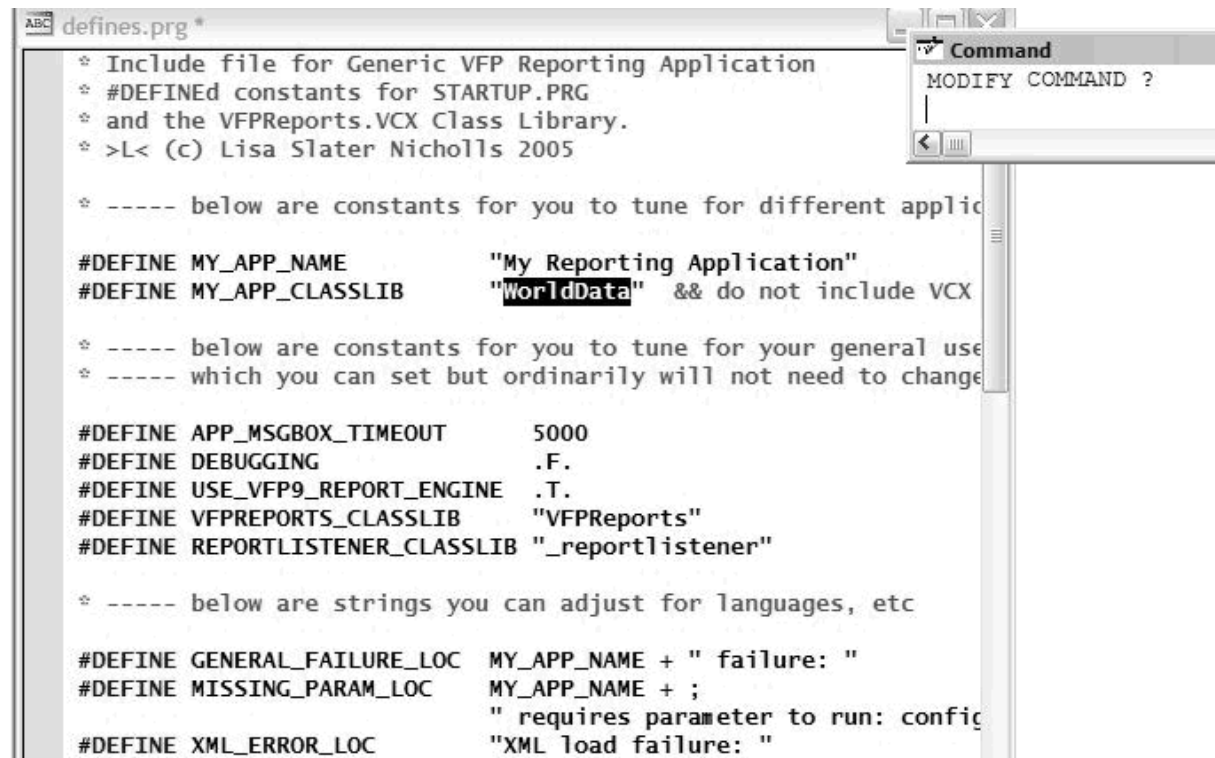


Figure 74. Customizing your VFP Application's constants

I've included two "make" files, or build scripts, to help you build the application: FIRSTBUILD.PRG and BUILD.PRG. You can see them in Figure 73 above. The FIRSTBUILD.PRG creates VFP project files in support of the build process, the first time you use the process for any application, so you'll use that one now.

Run FIRSTBUILD.PRG by choosing the **Program Do...** menu option, or simply by typing **DO ?** in the Command Window and selecting the FIRSTBUILD.PRG file. If you have already used the CD command to navigate to the "develop" folder, you can also type DO FIRSTBUILD in the Command Window.

When you execute the FIRSTBUILD program, you see VFP start the build process. As it identifies dependent files required by your application, it asks you to Locate any files it can't find by itself. For example, the UDF_GetDetails.PRG file is in the Reports directory, and I have included a reference to this program in DEFINES.PRG. VFP may also ask you to find the library you referenced in MY_APP_CLASSLIB, if it is not in the "develop" directory.

Press the **Locate** button in the dialogs that come up during the build process (see Figure 75), and find these files for VFP. You will have to do so twice for each of these files, because you are building two separate projects.

When FIRSTBUILD.PRG finishes running, it opens both projects for you (see Figure 76).