Programming Rubber Band Reports with Pivot Tables

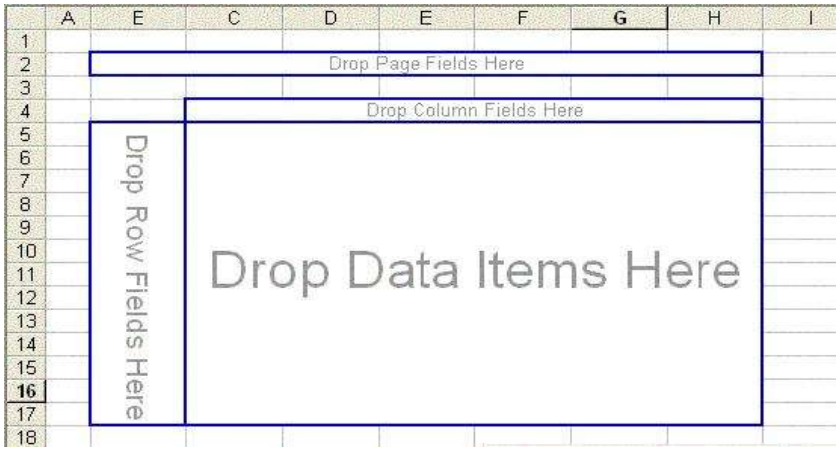
Hector Correa
hector@hectorcorrea.com

Introduction to Excel's Pivot Tables

Brief introduction to Excel's Pivot Tables

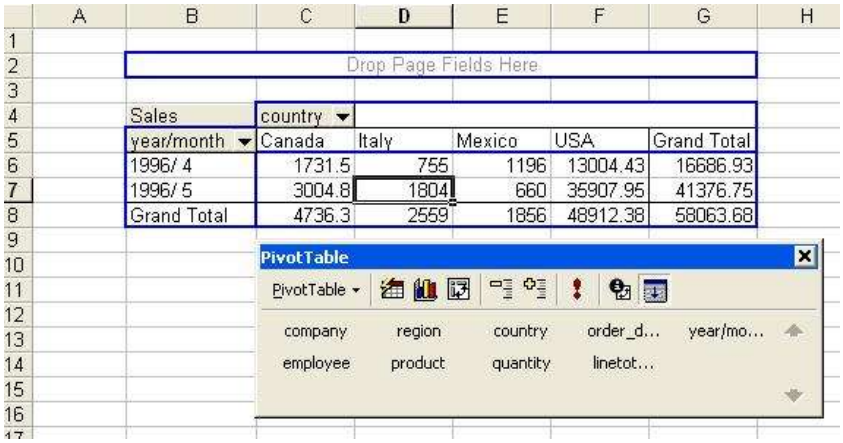
An Excel's pivot table is a special type of Excel table where you can rotate rows and columns to view data from different perspectives. The use of pivot tables is because they allow you to summarize large amounts of data by just dragging and dropping fields into the pivot table's rows and columns[1].

In its simplest form, a pivot table looks like the one depicted in the following picture. As you can see, a pivot table is composed of four basic elements: rows fields, columns fields, data items and page fields.



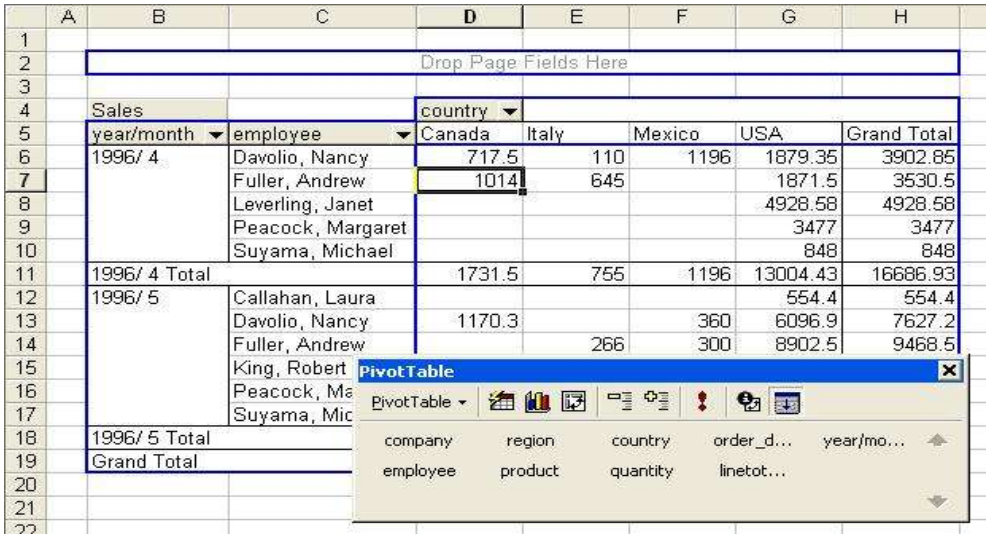
The following picture shows a pivot table with data coming from the TestData database that ships with VFP 7[2].

[1] Unless otherwise noted, the term *pivot table* in this document refers to Excel's pivot tables.
[2] Notice that this data is not the same as TasTrade database that is also shipped with VFP. TestData database contains the same information as Northwind database that ships with SQL Server.



In this example, you can see *sales* by *year/month* and *country*. Notice how fields in row and column headers are, in fact, drop-down controls. This means that when you click on them, a drop down list pops up enabling you to select the values that you want to show (filter) in the pivot table. In this particular case, I selected only the months of April and May in 1996 for four countries. A copy of the Excel file with this pivot table can be found in file PT_TESTDATA.XLS.

One of the most exciting features of pivot tables is that they are dynamic. This means that you can rearrange the fields displayed as rows, columns and data items to get different views of the same data. For example, the following picture shows the same data as the previous picture. However, in this case I dragged the *employee* field to the rows area. Notice how the pivot table automatically broke down information by employee.



Another interesting feature of pivot tables is their drill-down capability. You can double-click on any particular cell and Excel shows the specific records that are behind the summary

value displayed on the pivot table. For example, double-clicking on row *1996/4, Andrew Fuller* for column *Canada* shows details of the two sales that make up the 1014 dollars shon in the pivot table.

	A	B	C	D	E	F	G	H	I
1	company	region	country	order_date	yearmonth	employee	product	quantity	linetotal
2	Bottom-Dollar Markets	BC	Canada	4/13/1996	1996/ 4	Fuller, Andrew	Malaysian Coffee	9	414
3	Bottom-Dollar Markets	BC	Canada	4/13/1996	1996/ 4	Fuller, Andrew	Uncle Bob's Organic Dried Pears	20	600
4									

There are several other goodies that come along with pivot tables. For example, you can create pivot charts. These charts are associated with the pivot table and you can filter the information that is charted just as you can with the pivot table (i.e. via drop down lists). You can also create calculated fields (e.g. sums, counts, averages) in your pivot table. In addition, you can format your pivot tables to give them a professional look and feel by using a variety of built-in styles that come with Excel[3].

Even though pivot tables are very flexible and powerful, you might still wonder why you would look at Excel's capabilities to deliver your reports rather than using Visual FoxPro report writer or any of the third party reporting tools that are available. There are two strong reasons that make pivot tables a better candidate than other technologies to deliver reports to users.

First, most people already have Excel and know how to use it[4]. Allowing your users to create dynamic reports with a tool that they feel confident with is something that they will certainly appreciate.

Secondly, since pivot tables are so dynamic, users can customize their own reports by just dragging and dropping fields. They can drill down to any the level of detail that they need to by double-clicking on cells. Additionally, they can create charts by using one of the most well known chart engines available for end-users. The bottom line is that there is a significant reduction on the turn-around time to create reports and an increased flexibility in being able to represent and analyze data.

Programming Excel's Pivot Tables with Visual FoxPro

There are basically two ways of creating pivot tables in Excel. The first way is using Excel interactively. To do this, use the Pivot Table Wizard option that is on the Data pad in Excel's main menu. The second mechanism to create pivot tables is programmatically via OLE-Automation. This is the mechanism that I will describe in this document[5].

[3] There are plenty of books out there that describe how to use these and other features of pivot tables. In general, any decent book on Excel will have a chapter dedicated to pivot tables and pivot charts where you can find more information on how to format pivot tables for nice and elegant presentations.

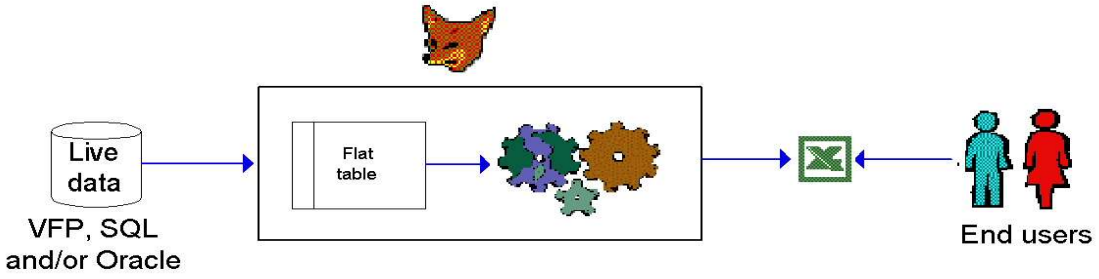
[4] Bellmer, Tom.

[5] Although I will not elaborate on the interactive way of creating pivot tables, I highly encourage you to give it a try before you dive into details on how to create them programmatically. This will be particularly useful if you have never played with pivot tables before. You can use the PT_TESTDATA.XLS file to do this exercise.

Before we dig into the details on how to create a pivot table programmatically, let us first take a look at the general process involved on creating a pivot table. This process can be divided into three basic steps:

1. First, you take some data stored in a normalized format and create a temporary table with a de-normalized (flat file) version of this information.
2. Second, you submit this de-normalized data to Excel and create a pivot table.
3. Third, once the pivot table has been created, you (or your end-users) only need to deal with the Excel file that you created. From that point on, you (or your end-users) do *not* need to be connected to either the normalized database or the de-normalized table.

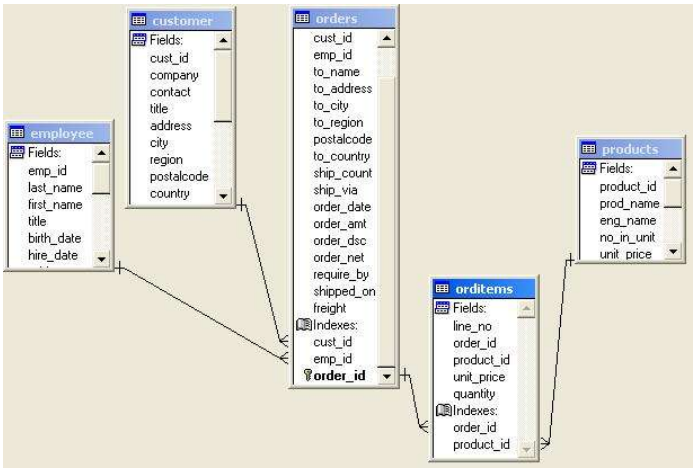
The following picture depicts this process:



Step 1. From normalized to de-normalized data

In well-designed systems, data is stored in normalized tables. In order to create a pivot table, you need to first create a de-normalized version of your database. Although this de-normalization might sound a little bit inefficient, the fact is that any reporting tool (including VFP's Report Writer) requires this de-normalized data to be created either implicitly or explicitly.

Let us say, for example, that you have a database like the one depicted in the next diagram. This is a diagram of the TestData database that ships with VFP 7.



The process to create a de-normalized table is usually pretty straightforward. You start by identifying *all fields* from *all tables* that you would like to include in the pivot table. For example, if you want to show information by customer, then you need to include the customer name (*customer.company*) in this de-normalized table. Likewise, if you want to display information by employee, then you need to include the name of the employee (*employee.first_name*) associated with each record. For the most part, you need to bring all descriptive names to the de-normalized table and omit primary keys.

Once you have identified what information you want to include in the pivot table, you then write a program in VFP to pull down this information and store it in a VFP table. The following fragment of code shows how you can create a de-normalized table for the tables depicted before[6]:

```
select      customer.company,       ;
            customer.region,        ;
            customer.country,       ;
            orders.order_date,      ;
            employee.last_name as employeelast, ;
            products.eng_name as "product",     ;
            orditems.quantity,      ;
            orditems.quantity * orditems.unit_price as linetotal;
from        customer                ;
            inner join orders on customer.cust_id = orders.cust_id ;
            inner join employee on orders.emp_id = employee.emp_id ;
            inner join orditems on orders.order_id = orditems.order_id ;
            inner join products on orditems.product_id = products.product_id ;
into table data\pt_testdata
```

The following picture shows a fragment of the data that will be generated with the previous SQL statement. Notice how fields like company name, employee and product fields are now de-normalized.

Pt_testdata

Company	Region	Country	Order_date	Employee	Quantity	Linetotal	Product
Ernst Handel		Austria	08/07/1993	King, Robert	5.000	75.0000	Flotemys Cream Cheese
La maison d'Asie		France	08/10/1993	Davolio, Nancy	40.000	280.0000	Sir Rodney's Scones
La maison d'Asie		France	08/10/1993	Davolio, Nancy	15.000	210.0000	Licorice
La maison d'Asie		France	08/10/1993	Davolio, Nancy	20.000	226.0000	White Chocolate
La maison d'Asie		France	08/10/1993	Davolio, Nancy	4.000	104.0000	Gramma Alice's Dumpling
Toms Spezialitäten		Germany	08/11/1993	Leverling, Janet	20.000	266.0000	Pickled Herring
Toms Spezialitäten		Germany	08/11/1993	Leverling, Janet	10.000	60.0000	Jack's New England Clam
Toms Spezialitäten		Germany	08/11/1993	Leverling, Janet	3.000	45.0000	Flotemys Cream Cheese
Rattlesnake Canyon Grocery	NM	USA	08/12/1993	Peacock, Margaret	10.000	320.0000	Malaysian Coffee
Rattlesnake Canyon Grocery	NM	USA	08/12/1993	Peacock, Margaret	15.000	343.5000	Perth Meat Pies
Rattlesnake Canyon Grocery	NM	USA	08/12/1993	Peacock, Margaret	60.000	1560.0000	Gramma Alice's Dumpling
Rattlesnake Canyon Grocery	NM	USA	08/14/1993	Peacock, Margaret	10.000	122.0000	Pavlova Meringue Desser
Rattlesnake Canyon Grocery	NM	USA	08/14/1993	Peacock, Margaret	2.000	63.8000	Rössle Sauerkraut
Morgenstern Gesundkost		Germany	08/17/1993	Buchanan, Steven	2.000	17.4000	Scottish Longbreads
Ernst Handel		Austria	08/18/1993	King, Robert	40.000	268.0000	Jack's New England Clam
Ernst Handel		Austria	08/18/1993	King, Robert	50.000	1150.0000	Pierrot Camembert
Ernst Handel		Austria	08/18/1993	King, Robert	40.000	928.0000	Wimmer's Delicious Brea

[6] Complete code can be found in the GETDATA.PRG file.

In the previous example, to create the de-normalized table a single SQL Select statement was required. In real life applications, however, the process can be more involved and require several SQL Select statements (potentially from several databases) plus a series of steps to massage the information until it has all the information that we will eventually need in the pivot table. Given VFP's data manipulation capabilities, it is easy to see why VFP is such a good fit for this particular duty.

Keep in mind that, although the de-normalized table will be a VFP table, this does not mean that the source data needs to be VFP data. You can use VFP to pull data from a variety of databases (e.g. SQL Server or Oracle) via SQL Pass Through or remote views.

There are some other routes that you can take to create this de-normalized table and some of them are even more powerful than the technique that is used in this document. A common mechanism to accomplish this task is via On Line Analytical Processing (OLAP) packages like the one that comes built-in in SQL Server. These mechanisms are beyond the scope of this document. If you are interested in them, you should take a look at Val Matison's white paper listed at the end of this document. Nevertheless, you should keep in mind that when you move to OLAP packages you not only increase pivot tables capabilities, but also the price and complexity of the solution.

Step 2. Creating the Pivot Table

Once you have your data de-normalized there are basically two methods that you can follow to create the pivot table.

One option is to copy your data to a sheet within an Excel workbook and then create the pivot table using this data. Although this approach works well in a majority of cases, it comes with a significant limitation: you can only import 65,000 records to an Excel spreadsheet. An example on how to create a pivot table using this approach can be found in the program, CREATEPT1.PRG

The other option that you can follow is to create a pivot table from an external data source (rather than from an Excel spreadsheet.) In this case, you ask Excel to read the data from a VFP table via ODBC or OLE-DB. Excel in turn reads this data to an internal cache (but it does not display it in a sheet) and then creates the pivot table. This is the technique that is used in this document.

It is very easy to create an Excel's pivot table from within VFP. Basically, you just need to use OLE-Automation to instruct Excel to create a pivot table for you. The following code fragment shows how to create a pivot table programmatically[7].

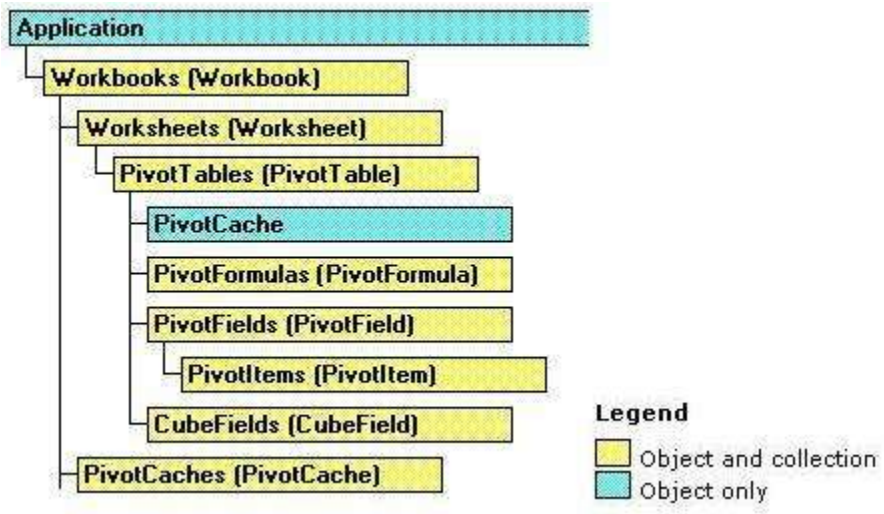
```
* 1. Launch Excel via OLE-Automation.
oExcel = createobject("excel.application")
oExcel.Application.Visible = .T.

* 2. Create a new workbook.
oWorkbook = oExcel.Workbooks.Add()

* 3. Define an Excel range object to dump the results into.
```

[7] Complete code for this example can be found in the CREATEPT2.PRG file.

```
oTargetSheet = oWorkbook.Sheets.Add()
oTargetRange = oTargetSheet.range("A2")

* 4. Define ODBC connection string and SQL statement
* that Excel will use to read data.
dimension aSource[2]
aSource[1] = "Driver={Microsoft Visual FoxPro Driver};" +;
             "SourceDB=" + DATAPATH + ";"                    +;
             "SourceType=DBF;"
aSource[2] = "select * from pt_testdata"

* 5. Create a pivot table object.
oPivotTable = oExcel.Sheets[1].PivotTableWizard( 2,;  && external data.
                        @aSource, ;
                        oTargetRange, ;
                        "PivotTable", .T., .T. )

* 6. Define how data would initially be arranged in the pivot table.
oPivotTable.PivotFields("country").orientation = 1    && row
oPivotTable.PivotFields("yearmonth").orientation = 2  && column
oPivotTable.PivotFields("linetotal").orientation = 4  && data
```

Notice how in step 5 we are using an Excel method called PivotTableWizard[8] to create the pivot table. Parameter aSource indicates to Excel to read data using the ODBC[9] connection and SQL statements that we defined in step 4.

Although pivot table wizard simplifies the process of creating a pivot table, it does not let you see how Excel organizes pivot tables internally. The following picture[10] shows the main objects used by Excel to handle pivot tables.



[8] Although this method is called PivotTableWizard, it is not a wizard that will prompt the user to follow certain steps. It is called a wizard because it automates some of the steps required to create the pivot table and shields the developer from some of the internal classes that Excel uses to create and handle the pivot table.
[9] PivotTableWizard only knows how to use ODBC to read data from external data sources. It does not support OLE-DB providers.
[10] Microsoft Excel Visual Basic Reference help file.

In order for Excel to create a pivot table, it first reads data to an internal (and invisible) object called PivotCache. The actual visible part of the pivot table is the PivotTable object. Inside the pivot table there are PivotFields objects (that represent fields stored in the data source) and PivotFormulas objects (that can be used to represent calculated fields.)

Let us look at an example on how to create a pivot table by directly manipulating Excel's pivot table and pivot cache objects rather than using the PivotTableWizard method[11].

```
* 1. Launch Excel via OLE-Automation.
oExcel = createobject("excel.application")
oExcel.Application.Visible = .T.

* 2. Create a new workbook.
oWorkbook = oExcel.Workbooks.Add()

* 3. Define an Excel range object to dump the results into.
oTargetSheet = oWorkbook.Sheets.Add()
oTargetRange = oTargetSheet.range("A2")

* 4. Create a pivot cache object.
oPivotCache = oWorkbook.PivotCaches.Add( 2 ) && external data

* 5. Tell this pivot cache the OLE-DB provider and SQL statement
* that Excel will use to read data.
oPivotCache.Connection = "OLEDB;Provider=vfpoledb.1;data source=" + DATAPATH
oPivotCache.Commandtext = "select * from pt_testdata"


* 6. Ask the pivot cache object to create a pivot table
* with the data.
oPivotTable = oPivotCache.CreatePivotTable( oTargetRange, "PivotTable" )

* 7. Define how data would initially be arranged in the pivot table.
oPivotTable.PivotFields("country").orientation = 1    && row
oPivotTable.PivotFields("yearmonth").orientation = 2  && column
oPivotTable.PivotFields("linetotal").orientation = 4  && data
```

The first three steps in this example are identical to the previous example. However, in steps 4 through 6 we are manually creating the pivot cache and pivot table objects rather than using the pivot table wizard. By doing this we now can ask Excel to read data using an OLE-DB provider rather than ODBC.

Although the use of an OLE-DB provider over an ODBC driver might not seem as an imminent advantage, it is certainly a much better strategy when you consider that this prevents Excel altogether from attempting to use Microsoft Query to read data into the PivotCache. Microsoft Query is an additional component that not all Excel users have installed and that is somehow difficult to configure programmatically[12].

[11] Complete code for this example can be found in the CREATEPT3.PRG file.
[12] Tamar, Granor and Martin, Della. Page. 189.

Step 3. Using the Pivot Table.

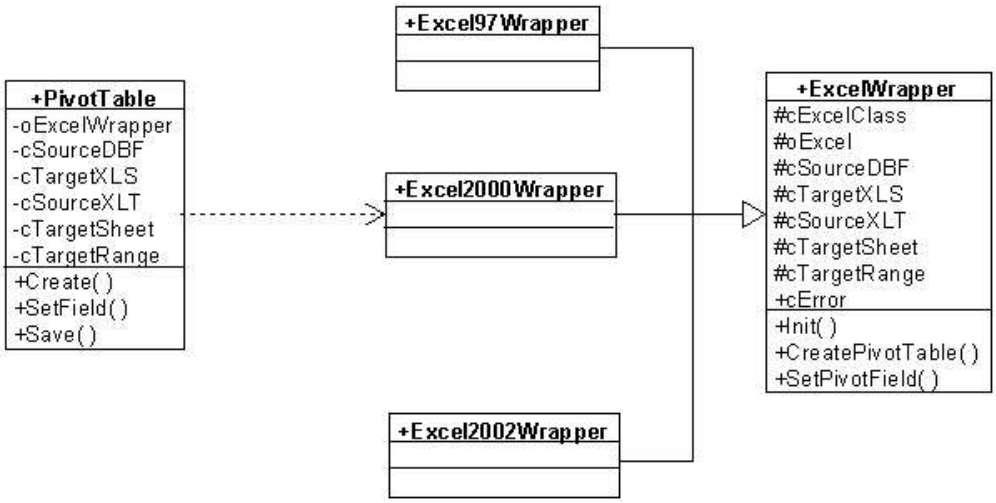
Once the pivot table has been created and stored in an Excel file, it is up to you to decide what you do with the Excel file. You can e-mail the Excel file to your users or post it to a shared location where they can access it. Remember that once the pivot table has been created, your users will not need your copy of the de-normalized data anymore since Excel saves its own copy of the data as part of the PivotCache object.

In addition, since data is stored internally in the Excel file, users still have drill-down capabilities when using the pivot table even though they are neither connected to the live database nor the de-normalized table! As you can see, deployment is reduced to delivering a single Excel file to the users.

You can add some extra functionality to the Excel file by using Visual Basic for Applications (VBA). For example, you can create forms (Microsoft Forms) inside Excel files with command buttons and checkboxes. This topic is out of the scope of this document, but there are plenty of books on this topic.

Paving the way

Included with this white paper, there is a set of classes[13] that encapsulate access to Excel's pivot tables. The following picture shows the class hierarchy of these classes. The only class that you need to instanciate is PivotTable. This class will automatically call the other classes.



The following fragment of code shows how to use the PivotTable class to create a pivot table for a VFP table called PT_TESTDATA.DBF

```
oPT = createobject("pivottable" )
oPT.cSourceDBF = "c:\efox_pivot\data\pt_testdata.dbf"
oPT.cTargetXLS = "c:\efox_pivot\data\pt_testdata.xls"
```

[13] Code for these classes can be found in the PTCLASS.PRG file. Samples of usage can be found in the PTSAMPLES.PRG file.

```
oPT.Create()
oPT.SetField( "country", "row" )
oPT.SetField( "yearmonth", "column" )
oPT.SetField( "linetotal", "data" )
oPT.Save()
```

As you can see, this class simplifies the process of creating the pivot table quite a lot since now you don't need to deal with Excel directly, but rather with a much simpler interface. Source code for this class is provided so you can extend it and adapt it to your particular needs.

Excel's Pivot Tables and Your Users

Advantages of Using Pivot Tables

Pivot tables are a powerful tool that you can provide to your users to help them analyze data from multiple angles. One of the benefits of them is that, since most users are familiar with Excel, training time can be reduced dramatically without sacrificing flexibility.

An additional advantage of using pivot tables is the reduction on the number of hits to the live database by letting users query information from an offline data set. Remote and mobile users would also appreciate offline capabilities.

Finally, by giving your users such a flexible tool, you may reduce the number of requests for new reports that your users will make. With pivot tables, users can easily create these reports, and even charts, themselves by simply dragging fields to rows and columns. It does not get any easier than that.

References

- Bellmer, Tom. *Using VFP 6 and Excel 2000 as a Reporting Tool*. Paper delivered at the Midwest FoxPros Users Group (May/2001.) http://visionds.net/hcorrea
- Granor, Tamar and Martin, Della. *Microsoft Office Automation with Visual FoxPro*. Hentzenwerke Publishing. 2000.
- Mic*rosoft Excel Help*. File XLMAIN9.CHM that comes with MS Office 2000.
- *Microsoft Excel Visual Basic Reference*. File VBAXL9.CHM that comes with MS Office 2000.
- Stearns, Dave. *Programming Microsoft Office 2000 Web Components*. Microsoft Press. 1999.
- Walkenbach, John. *Excel 2002 Power Programming with VBA*. Hungry Minds, Inc. 2001.
- Walkenbach, John. *Microsoft Excel 2000 Power Programming with VBA*. Hungry Minds, Inc. 1999.
- Wells, Eric. *Advanced Spreadsheet Programming with Microsoft Excel 97*. Microsoft Office Developer Web Forum. 1998. http://www.eu.microsoft.com/exceldev/artcles/movs104.htm