Developing REALBasic Database Applications

by

Kevin Cully

Presented at
FoxForward Conference
September, 2007

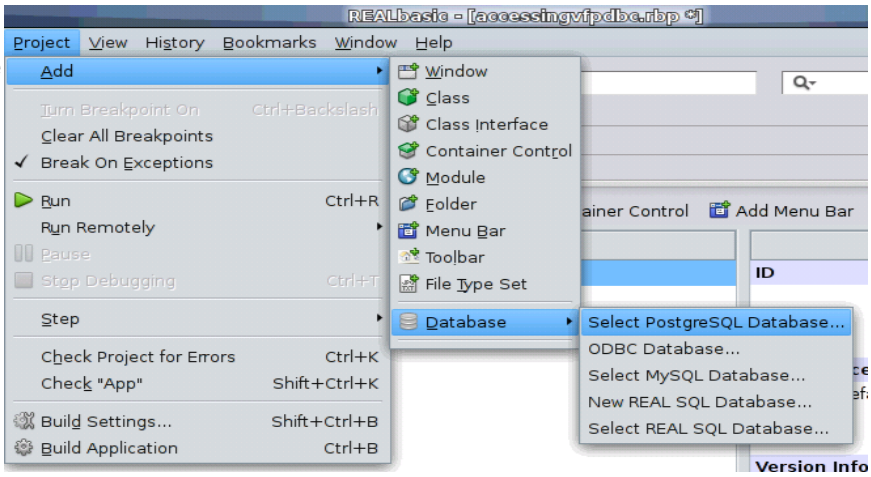
Introduction

I'm writing up this paper in the hopes that FoxForward attendees that chose not to attend my session(s) would still be able to take away some knowledge of REALbasic (RB). REALbasic really is worthy of a look as a potential development tools for Visual FoxPro developers. While I have this paper available, the presentation won't just be a read-through of the paper.

There is so much to show, there is just no way that I could address even a small subset of the features of RB. I'll be focusing on building database applications but even with that limitation, the presentation will be very shallow on the details.

Database Plugins

My application will be accessing PostgreSQL data. Here you can see some of the databse options available under the Professional Edition of RB and how to add them into the project.



Accessing VFP Data From REALbasic

I know that people will be asking about whether you can access VFP data from RB and the answer is that you can ... just not very well. It is easy enough to get a record set from a DBF, but you can't query against it in the manner that we have become accustomed. In fact, the RecordCount property of the record set always returns -1 which is a minor annoyance.

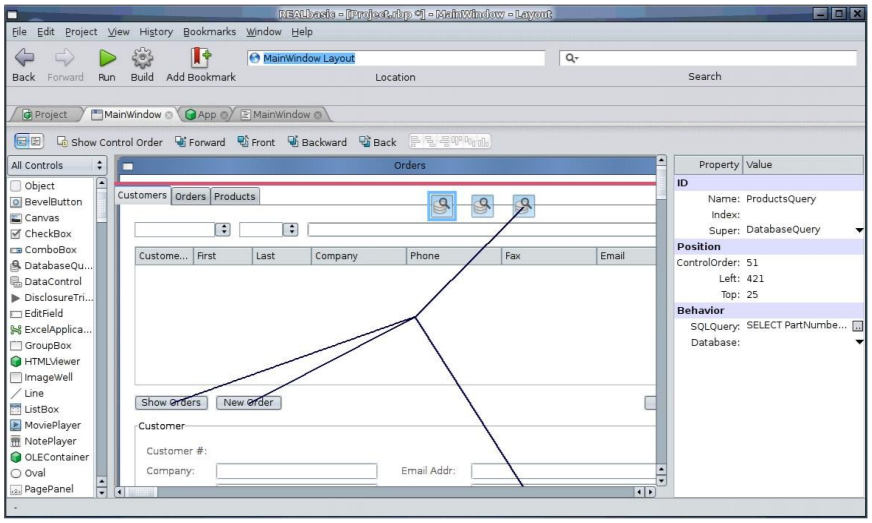
I'm sure that more capabilities are available in RB on the Windows platform, but that defeats one of the greatest strengths of RB. The real goal would be to be able to do a query against VFP data on the Linux or Mac platforms. There still may be a way to get this accomplished, I just haven't found it.

```
DIM rs as new DatabaseCursor
DIM datalocation as new FolderItem
DIM lnX AS Integer
DIM cMsg AS String

datalocation = GetFolderItem("LargeData.DBF")
rs = OpenDBFCursor(datalocation)

IF rs = nil THEN
  msgbox "Database missing"
else
  rs.MoveFirst
  msgbox str(rs.RecordCount) + " records  "+ _
      str(rs.FieldCount) + " fields"
  FOR lnX = 1 TO 5
    cMsg = rs.Field("Cname")  // Name of one of the fields in the table.
    msgBox cMsg
    rs.MoveNext
  NEXT
END IF
```

The No-Code and the Coded Database

RB has a built in DataControl class that attempts to make data access easier in RB. It assists in binding the fields and the values of the data. This is similar to setting a ControlSource and having the table in the DataEnvironment in VFP. The DataControl is bound to a particular table, and this table can be coming from any database that can be connected to. When a query is executed using the DataControl.SQLQuery property to hold the query, the DataControl.RecordSet property is populated with the results of the query. You can probably guess that the DataControl has additional methods to handle the typical data manipulation operations such as NewRecord, Insert, Update, and Delete.

There is also the DatabaseQuery class. It does just what it's name says and holds a query in its SQLQuery property. This class makes it easier to see a list of the results from the query. This class can be bound to buttons and display objects on the form, all for a no-code way of binding the data and the display of that data.



All is not as it seems however. Just like certain approaches to handling data in VFP will have advantages and disadvantages, so does the no-code approach to handling data access. Yes, you gain simplicity, but you give up some control. It is generally regarded that veteran RB developers avoid the no-code approach because of the loss of control.

For the rest of the paper and for the presentation, I'll be demonstrating the coded approach. This is actually the approach that I have taken with VFP as well. I rarely use control sources, the data environment, views and so on. Perhaps this eases my transition to RB, where others will dread the extra labor to accomplish a similar result. I find that the extra level of control and error handling is of greater appeal than to leave it up to a certain amount of “magic” in the development process.

Global Object, Global Connection Management

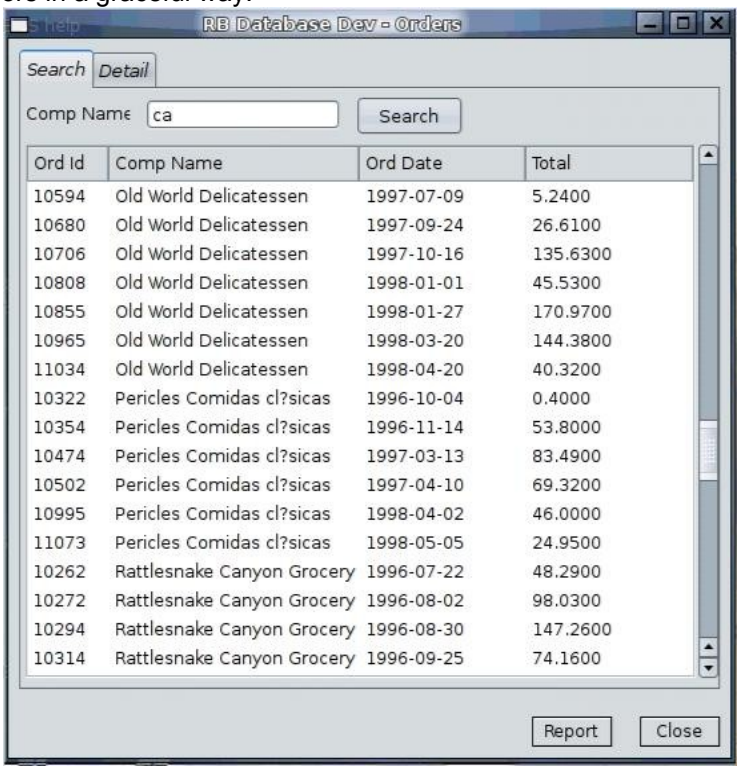
It is also commonly understood that connections to the database should be utilized and terminated in RB. In VFP, I could open a connection to a SQL Server and hold it open for hours on end. In RB, this connection will time out, and it is not at all easy to exactly determine the state of that connection. The correctly held belief is that you connect to the database, accomplish your task, and then disconnect until the next operation.

So, with all of this work needing to be done, is there a way to make this easier? Of course. We can set up a global connection manager and have it take care of connecting, executing, disconnecting, some error handling, logging, and anything else we need it to do.

Already, our application has a global application object named APP. We can call it from anywhere in the application and have it do our bidding. The global application gets a couple of properties and methods. Some are private, while others are globally accessible.

To the application object we add a property of “db” to hold the PostgreSQLDatabase object while connected. There is also a “rs” property that holds the RecordSet that are the results of the queries made against our database. This RecordSet is passed back to the calling method and consumed there. We also have a “rsUser” property that holds the RecordSet of the user that has logged in. We will use this all over the place so we'll just keep that handy.

The custom methods on the global application object that are most important is the “SPTExecute” and the “SPTSelect” methods. We use the “SPTSelect” method when we want to get a RecordSet back from the database as in the case where we're making a SQL Select statement. The “SPTExecute” method is called when we are not expecting a RecordSet back from a database operation. Examples are SQL Inserts, Updates, and Deletes. Both of these methods take care of connecting and disconnecting from the database for us. They also handle database errors in a graceful way.



Query, Insert, Update, and Delete

As you can guess by the name of the methods, we're using a SQL Passthrough methodology to access the data.

From the login screen, you can see how we are assembling a SQL statement.

```
Function ChkLogin(cUserID AS String, cPassword As String) As Boolean
  DIM cCommand as String, lResult As Boolean=FALSE

  cCommand = "SELECT * FROM " + App.cSchema + ".users WHERE UPPER(cUserID) LIKE '" + _
      UPPERCASE(cUserID) + "%' AND UPPER(cPassword) LIKE '" + _
      UPPERCASE(cPassword) + "%'"
  App.rsUser = App.SPTSelect(cCommand)

IF NOT App.rsUser.EOF THEN
    OrdersWindow.Show
    Self.Close
    lResult=TRUE
  Else
    IF Me.nAttempts < 3 THEN
      MsgBox("The user name and password is not correct.  Please re-enter")
      Me.nAttempts = Me.nAttempts + 1
    ELSE
      MsgBox("You have exceeded the number of failed login attempts.  Closing.")
      Me.Close
    ENd if
  End IF

  RETURN lResult
```

Showing Data

Once our data has been retrieved from the database, we typically have to display lists, or the detailed data. We're doing our database display in a coded way, so we set the value of each control. This will seem like a lot of work to many of you, but when you consider having to set each control source, this is just a different manner of work.

```
Function OrderQuery() As Boolean
  dim lReturn as boolean
  dim cQuery as String
  dim nRecNo as Integer

  lReturn = False

  cQuery = "select orderid, companyname, orderdate, freight from ff07.orders2 where 1=1"
IF txtSearchComp.Text <> "" THEN
    cQuery = cQuery + " and LOWER(companyname) like '%" + txtSearchComp.Text.LowerCase + "%'  "
END IF
  cQuery = cQuery + " order by companyname"
  rsOrder = APP.SPTSelect(cQuery)

  lstOrders.DeleteAllRows  //empty the listbox
IF rsOrder.eof = True THEN
    MsgBox "Your query returned no records.  Please adjust your criteria and press 'Query' again."
ELSE
  do until rsOrder.eof  //loop through record set, stored in myrec, until it reaches EOF (end of file)
      lstOrders.addrow ""  //adds an empty row
      nRecNo=lstOrders.listcount -1  //sets y to the listcount minus 1, which is the total number of rows...this
      lstOrders.cell(nRecNo,0) = rsOrder.Field("orderid").stringValue  //sets mylistbox cell (y,0) to the id field
      lstOrders.cell(nRecNo,1) = rsOrder.field("companyname").stringValue
      lstOrders.cell(nRecNo,2) = rsOrder.field("orderdate").stringValue
      lstOrders.cell(nRecNo,3) = rsOrder.field("freight").stringValue
      rsOrder.movenext  //moves to the next record in the recordset
    loop
END IF
  Return lReturn
```

Retrieving Data From Form Controls

The retrieving of the data is just the opposite of the displaying of the data. The data needs to be retrieved and manipulated into a SQL statement that can be passed back through to the database.

```
Function OrderUpdate() As Boolean
  dim lReturn AS Boolean
  dim cSPT AS String
  dim cOrderID, cCompanyName, cOrderDate, cFreight, cAddress, cShipName AS String

  lReturn = False
  cOrderID     = txtOrderID.Text
  cCompanyName = txtCompanyName.Text
  cCompanyName = ReplaceAll(cCompanyName, "'", "\'")
  cOrderDate   = ReplaceAll(txtOrderDate.Text, "'", "\'")
  cFreight     = ReplaceAll(txtFreight.Text, "'", "\'")
  cAddress     = ReplaceAll(txtAddress.Text, "'", "\'")
  cShipName    = ReplaceAll(txtShipName.Text, "'", "\'")

  // EditSpecies method on the SpeciesEdit form returns false if the user cancels
  cSPT = "UPDATE ff07.Orders2 SET " + _
      "companyname='" + cCompanyName + "', " + _
      "orderdate='" + cOrderDate + "', " + _
      "freight='" + cFreight + "', " + _
      "address='" + cAddress + "', " + _
      "shipname='" + cShipName + "' " + _
      " where orderid = " + cOrderID

  lReturn = App.sptExecute(cSPT)
  IF lReturn THEN
    MsgBox "Your record has been updated."
  END IF

  RETURN lReturn
```

Reporting

Reporting in RB is one of it's weaknesses. To get the capabilities of the VFP report writer, you'll have to purchase a add on product, namely the report writer from On-Target Reports.

Of course we can get inventive with the report writer as well. Two that are feasible, cross platform is (1) the creation of HTML based reports with the navigation to the output file using a browser either inside or outside of the RB application and (2) the use of the cross-platform open-sourced TeK and LaTeK tool kit. LaTeK can output to PDF files using the TeKtoPDF toolkit. I've done some work with LaTeK in VFP and this would be a great solution that can create high quality output with lots of flexibility.

In the code to the right, I've created a report using a HTML file, and launching the default browser with the file loaded.

```
Sub PrintOrder()
  DIM oFolderItem AS FolderItem
  DIM nBytesWritten AS Integer = -1
  DIM cTemplate AS String = ""
  DIM cOrderID, cCompanyName, cAddress, cFreight AS String

  cOrderID = rsOrder.Field("orderid").StringValue
  cCompanyName = rsOrder.Field("companyname").StringValue
  cAddress = rsOrder.Field("address").StringValue
  cFreight = rsOrder.Field("freight").StringValue

  cTemplate = App.FileToStr("dbrtemplate.html")
  cTemplate = ReplaceAll(cTemplate, "<%OrderID%>", cOrderID)
  cTemplate = ReplaceAll(cTemplate, "<%CompanyName%>", cCompanyName)
  cTemplate = ReplaceAll(cTemplate, "<%Address%>", cAddress)
  cTemplate = ReplaceAll(cTemplate, "<%Freight%>", cFreight)

  nBytesWritten = App.StrToFile("databasedev.html", cTemplate)

  IF nBytesWritten > 0 THEN
    oFolderItem = getFolderItem("databasedev.html")
    oFolderItem.Launch
  END IF
```

Building For Multiple Platforms

We'll look at the settings for building the application cross platform.

RB takes the majority of the cross platform work out of the development process. Not everything is perfect from one platform to another. As an example, there is a large number of complaints of developers moving from Windows to the Linux platform based on the size of the fonts in Linux, where they don't match up with the Windows platform. Certain developers will detect the platform and adjust the font size properties of the controls. Other developers find that developing separate forms per platform is the approach that gets them the results that they desire. My approach is to develop for the Linux platform and the results for the Windows platform is mostly acceptable in my opinion.

Conclusion

I hope that I find that RB is an exciting, easy to get started, viable development environment. Of course, the cross platform development features are absolutely killer. In addition to that, REAL Software has the utmost interest and excitement for the future and continued enhancement of the RB product.

From a professional perspective, the ability to differentiate myself from the majority of other database developers in the ability to be able to target the Mac and Linux platforms, which I believe are two emerging business platforms.