

VBA! Will It Ever Die?

Shôn Ellerton, June 24, 2019

Do you feel like a dinosaur or get condescending looks from programmers when you state you're an expert with VBA? So why is it that job profiles are increasingly seeking VBA experts? Will VBA ever become a deprecated feature of MS Office?



VBA! Will it ever die?

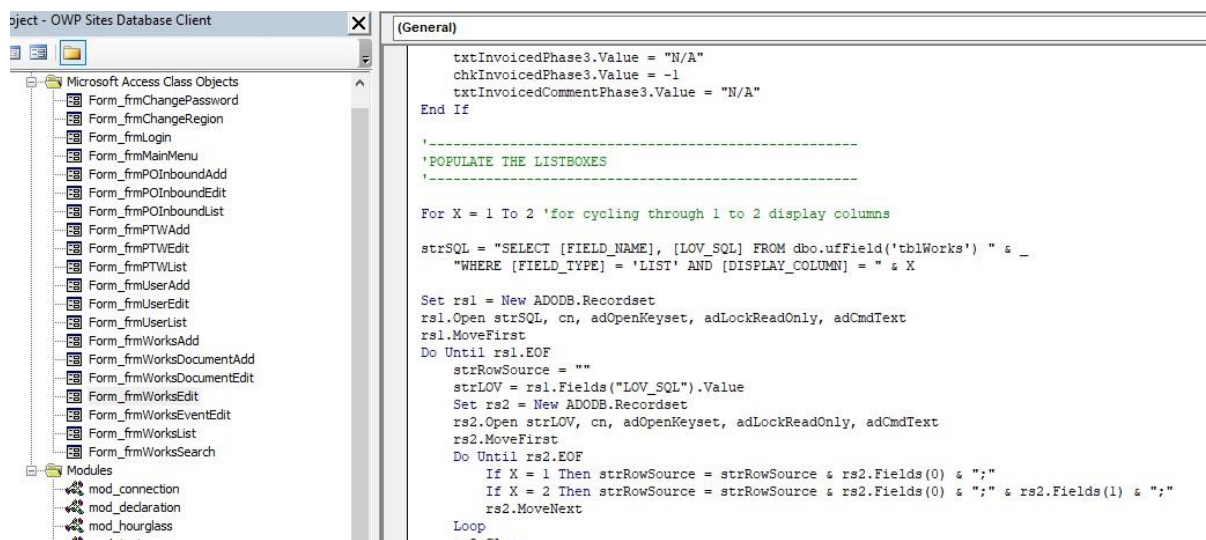
I recently came across a job profile that required someone with expert knowledge of VBA (Visual Basic for Applications) to maintain *and improve* their legacy project-management systems. It sparked my interest because I consider myself to be an expert in VBA having programmed with it since 1997 creating a wide assortment of MS Access and Excel client applications which were very successful during their times. Until recently, I deliberately *avoided* mentioning VBA on my CV should I be considered a dinosaur within the IT industry. However, that now appears to be a mistake.

VBA is quite a powerful procedural programming language in-built into Microsoft's Office products since the mid-1990s. Best known for its macros, VBA can do so much more. You can build complete applications using any of Microsoft's Office products which is incredibly attractive for many businesses running Microsoft Office as no additional licencing costs are needed as the software has already been installed.

Like COBOL and FORTRAN, VBA is here to stay for a very long time despite the near-extinction of dedicated VBA training courses offered by popular online training platforms and the decrease in numbers of those proficient in programming it.

What is VBA anyway?

If you do not know what VBA is, it stands for Visual Basic for Applications. VBA sits at the heart of our everyday Microsoft Office products including Word, Excel, Access, Visio and many more. You can create simple macros up to highly advanced applications by creating independent modules and classes importing a wide variety of reference libraries including those from other Office products, third-party products and, of course powerful data libraries like ActiveX Data Objects (ADO). Below is a screenshot of a part of a procedure I wrote in VBA accessing data using ADO to populate drop-down boxes within an Access client.



```
txtInvoicedPhase3.Value = "N/A"
chkInvoicedPhase3.Value = -1
txtInvoicedCommentPhase3.Value = "N/A"
End If

'-----
'POPULATE THE LISTBOXES
'-----

For X = 1 To 2 'for cycling through 1 to 2 display columns

strSQL = "SELECT [FIELD_NAME], [LOV_SQL] FROM dbo.ufField('tblWorks') " & _
"WHERE [FIELD_TYPE] = 'LIST' AND [DISPLAY_COLUMN] = " & X

Set rs1 = New ADODB.Recordset
rs1.Open strSQL, cn, adOpenKeyset, adLockReadOnly, adCmdText
rs1.MoveFirst
Do Until rs1.EOF
    strRowSource = ""
    strLOV = rs1.Fields("LOV_SQL").Value
    Set rs2 = New ADODB.Recordset
    rs2.Open strLOV, cn, adOpenKeyset, adLockReadOnly, adCmdText
    rs2.MoveFirst
    Do Until rs2.EOF
        If X = 1 Then strRowSource = strRowSource & rs2.Fields(0) & ";"
        If X = 2 Then strRowSource = strRowSource & rs2.Fields(0) & ";" & rs2.Fields(1) & ";"
        rs2.MoveNext
    Loop
Loop
```

One can build independent client applications simply by wrapping them into a VB6 project. Although VB6 was deprecated in 2008, VBA still lives on and was even given a bit of an upgrade in 2010. Microsoft's current suite of *dot*.NET projects (C#, VB, C++) offer significantly more in the way of features and functionality; however, the steeper learning curve of building and deploying .NET applications is very much largely in the domain of the dedicated programmer whereas getting started with VBA is exceptionally easy. There does come a point, when building complex applications using VBA rather than .NET starts to become more difficult as one needs to delve into complex API calls and COM objects. Historically, before the rise of the dedicated professional programmer, all computer programs were written by those directly in the business for which the program was required. For example, my late grandfather wrote countless number of FORTRAN programs to run data analysis results for sugar beet trials back in the 1960s.

My story with VBA

As for my own story, I've been working with data for more than 20 years and managed to build entire client/server database systems for a wide variety of projects, especially within the telecommunications and wireless networks industries. Many of these systems, I designed from first principles taking requirements from the project's consumers, building a suitable relational data model, providing a means for users to access and update data, supplying a reporting platform and implementing automated data exchange routines between the client and the project. In a nutshell, the primary object of the game was to stop the practice of using spreadsheets to *store* corporate and project data and, instead, migrate the data to a single-source of truth to which spreadsheets could *consume* the data for analysis and reporting use.

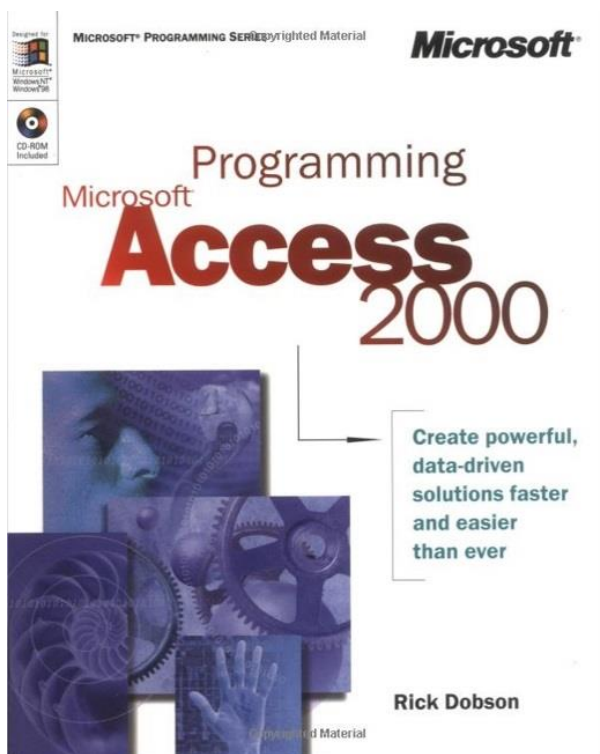
Many of these systems grew into full-blown client/server systems using a variety of tools including MS SQL Server, SSIS, SSRS, SSAS, C#-coded web services, Power BI, and of course, the humble Excel spreadsheet to consume data. Not surprisingly, deploying blank 'refreshable' Excel spreadsheets with direct ODBC connections has always proved to be *very* popular with project teams, usually much to the dislike of the IT departments due to security considerations and a variety of other reasons. Whilst working with Visionstream on a national Optus mobile base station rollout project in Australia, these refreshable spreadsheets constituted 90 percent of all user reporting. Let me repeat that, 90 percent! For nearly five years during the project, the IT department kept threatening to close off the direct connections but having offered no other practical solution, project leaders wielded their sticks and objected and the threat never materialised. This battle is commonplace across many industries.

Where VBA came useful was when I had the challenge of creating a multiuser database environment using MS Access as the client and MS SQL Server as the database. It was easy to link tables directly and create basic forms with Access; however, the system got overwhelmed very quickly and died as soon as you had more than a dozen users using it where large datasets needed to be accessed. To remedy the situation, I used Access as a thin-client application with no local data, purely using the underlying VBA/ADO code to access datasets from the remote database. The application was 'stateless' implying that connections were only opened when needed but then immediately closed when not needed. Only the relevant data queried was carried to the client which meant bandwidth requirements were minimal, an important aspect particularly in the days of slower networks. 'Write' tasks were always executed via a stored procedure,

never directly against a base table, so security was never an issue so long as the credentials were specified correctly. Another great feature about these clients are that they are lightning fast so long as you code sensibly. Although web-based clients can be pretty nimble and quick these days, they still have far more latency than desktop client applications thus making the user experience far more slow and cumbersome.

Coming from a civil engineering background, I had no formal training in programming apart from a little FORTRAN. Not satisfied with how MS Access functioned without customising it back in 1999, I purchased the marvellous relic of a book shown below (along with a sample page) which set me on the path of understanding recordsets and connectivity to datasources using VBA with ADO. To me, at the time, it was the bible of how this all worked, and even now, it explains the principles thoroughly, although the Internet has everything I need these days.

One thing for sure, I worked way too many late nights and weekends in single-handedly delivering these systems!



```
Copyrighted Material
Programming Microsoft Access 2000

'Reference connection.
Set cnn1 = CurrentProject.Connection

'Execute SQL for maketable query.
cnn1.Execute "SELECT FamilyMembers.* INTO " & _
"FMBBackup FROM FamilyMembers"

'This procedure fails if FMBBackup already exists.
End Sub

Unfortunately, there is generally more to creating and running a
SELECT...INTO statement than these two ADO statements. For example, this
procedure can fail if the FMBBackup table already exists. It can also fail if an-
other user opens either table. These and other complications demand error
trapping. The following sample illustrates how you might start error process-
ing for an application that uses a procedure with a SELECT...INTO statement.

Sub MyMakeTable2()
On Error GoTo Make2Err:
Dim cnn1 As ADODB.Connection

'Reference connection.
Set cnn1 = CurrentProject.Connection

'Test for unanticipated errors.
Err.Raise 1

'Execute SQL for make-table query.
cnn1.Execute "SELECT FamilyMembers.* INTO " & _
"FMBBackup FROM FamilyMembers"

Make2Exit:
'Close the connection and set it equal to nothing
and exit the sub.
cnn1.Close
Set cnn1 = Nothing
Exit Sub

Make2Err:
'Trap for table already exists.
If Err.Number = -2147217900 Then
cnn1.Execute "Drop Table FMBBackup"
Resume
Else
MsgBox "The program generated an unanticipated " & _
"error. Its number and description are: " & _
```


VBA today

It so happens that many millions of applications have been written worldwide using Microsoft Access with clever and creative VBA coding. Moreover, many of these applications are still in existence today. And this is now turning into a bit of a problem.

For anyone who has deployed Microsoft Office applications with custom VBA code may have experienced the exasperating problem of Microsoft's ongoing product evolution, much of which, involves 'death by a thousand cuts' by slowly deprecating features required to run the older applications. I experienced this with a MS Access application called an Access Data Project (ADP) which is now not at all runnable on the latest MS Access package. I once had to convert one of the applications I made in the past for use in a present system by running an older version of MS Access and converting it to the newer ACCDB format, not a trivial matter if you don't know how to. However, and this is the beautiful thing, the VBA code still works flawlessly once the database connection code was altered to suit.

I have noticed an increase in the number of job adverts looking for those who are familiar with VBA coding, particularly with respect to database-driven clients using MS Access. I encountered one the other day from an Australian mining company who have a large quantity of legacy applications built like the ones I used to build and now want to migrate this functionality to web-based solutions. The last two applications I built (both for telco projects) were both converted to web-based solutions; however, this required a considerable amount of resources, and hence, expense, to convert and build. In an Agile environment, this process required a team of specialists including a lead developer, C#.NET programmers, a scrum master, an overseas team, a deployment tester, a trainer, a DBA, web designers, and of course, the creator of the original project during the initial stages. Moreover, the processes for changing the application became far more exacting, procedural and costly. Many businesses do not have this luxury and continue to use their legacy applications. However, as mentioned earlier, the risk of losing required functionality by feature deprecation is a real risk.

The biggest challenge businesses have using these legacy VBA-based systems is ensuring that everyone has a version of the Office product which works. Moreover, any changes to the client must be distributed out to the users, whereas in a web-based environment, this is not an issue so long as the user's browser is compatible.

Conclusion

The question remains how long Microsoft will continue to support VBA in its suite of Office applications. There are rumours that Javascript or some other scripting language might take the place of VBA, or at least, run along beside it as an optional scripting tool. Whether Javascript is better than VBA is out of scope in this article; however, it would be safe to say that deprecating VBA within the near future will encounter very strong opposition simply due to the number of applications still using it worldwide.

Those embarking on learning computer languages may not regard learning VBA to be of future significance to their IT careers; however, I believe VBA will still be extensively used for many years ahead.

As for learning legacy programming languages, look at what's happening with COBOL programmers. There are many very large COBOL systems still running today which require ongoing support. Experts in the field are getting harder to find, most of which are either retired or passed away. Interestingly, many Indian IT learning establishments have capitalised on this by offering COBOL as a programming elective.

If you know VBA well, show it off. Like any programming language, what is important is learning the principles of good programming like logic, efficiency and code re-use. Don't let anyone tell you that VBA isn't a proper programming language and that it's on its dying legs. It's likely that whoever would say that never did any programming with VBA!