Access 2007 & the Ribbon – It's really no big deal

OverView:

Once in a major news group I was worrying out loud about something and the head guru replied, publicly, "Woody, you don't need to worry about that. With a name like Woody, nobody ever took you serious in the first place." That was twenty years ago, but take that as the caveat for what I have to say below. <grin>

I have been working full time with Access 2007 for two years now and have noticed something significant. Many, if not most, Access developers really don't want to move to the newer version, and I think one of the main reasons has to do with the new Ribbons associated with Office 2007. But, It's really no big deal.

I will give the short version of my logic here, and talk about it in broader terms later, but the main reason for switching to 2007 has to do with Microsoft itself. There are big fish and little fish. The little fish "**make it**" off the back of the big fish. We are little fish. If we separate ourselves from the big fish, we have no future, short of attaching to another big fish. The only other big fish out there (that I care about) is .NET. .Net is a big ugly whale with giant huge spouts of water coming out its snout and, though pretty to watch, not something this little fish really wants to attach to. But Access I can live with. I like it. I enjoy it. It is fun to use. I'm not going to sacrifice my career over my refusal to understand ribbons. Microsoft is not going to go backwards. Menus are dead and a thing of the past. Ribbons, or their successor, are the things of the future for all of Microsoft Office; at least, this is my opinion. I get an opinion—Right?

The second reason for getting into Ribbon logic is that it really isn't that hard. Now I should qualify that. It is hard and requires a great deal of knowledge if you're going to write applications that look like they come from New York. But, if that's not the case, it's no big deal. It may take a week. But, would you sacrifice your career for a weeks worth of new knowledge. If you're sharper than I am it may take you considerably less time, but lets go with the one week scenario for now.

The reason it's going to take a week is not because it's difficult, but because it's massive, and you have to first understand the big picture of what it's designed to do, even if all you're going to do is make use of 10% of its features. I will get into more detail shortly, but for the moment there are basically four things you need to understand and do.

1). **Understand the UsysRibbons table** and how it interacts with Access.
2). **Understand how to link a form or report to a custom ribbon.**
    This is this is ridiculously easy. You fill in a name in a property of your form or report.
3). **Understand the syntax for the RibbonX xml**:
    This is really the hardest part but the good news is that once you figure it out for one custom ribbon, you simply copy and paste for all others.
4). Understand Call-Backs:
    If you like VBA, this is enjoyable. It's just a matter of telling Access what to do when someone selects a certain object in the Ribbon.

OverView Summary:

So that's the overview. The details follow. I'm not going to go into detail with a step by step approach, the books will do that; and I'll point you to the books. Rather, I'll give you just enough to take the edge off and help you not to feel intimidated by custom ribbons.

And Finally:

This document may be read on its own but it is really meant to be read in conjunction with three other files. They are:

RibbonTemplate_02.accdb
rbnRibbonDemo_01.xml
rbnReport.xml

RibbonTemplate_02.accdb is a small database I created for the purposes of showing off this logic. It is self contained and has everything in it that is talked about here

rnbRibbonDemo_01.xml and rbnReport.xml are the actual XML files that make the ribbon part of the demo database work. These may be opened in an XML editor (like Bonfire) or in regular ole Notepad.

Use this document as a guide to understanding the three files mentioned above. It's really not very hard.

How to get started:

I don't know how to tell *you* to get started, but I'll tell you how **I** did. I have learned that you need at least three good (but different) books to master any one expansive subject. Prior to Access I programmed for seven years in Paradox For Windows, seven years in PowerBuilder, and three years in .Net. I had no one to teach me so I had to learn for myself. When I switched to Access I bought the best three Access 2007 books I could find. Each had their strengths and weaknesses. Two of them did not help me much where Ribbons are concerned, but the third one did. It is called **Access 2007 Inside and Out by Microsoft Press**. There were two particular chapters that helped; Chapter 23 and 24. These chapters are clear and lucid and if you follow their examples one step at a time, you will arrive at where you want to be. If you don't have something better, I suggest this book. Just take it slow and easy, one step at a time. I took more time than I really needed because I wanted to understand things that had to do with the Ribbon and SQL Server, but later I dropped that and just stuck with regular old Access (.accdb). If you really feel you need SQL Server as the back-end, maybe it is indeed time to switch to .NET.

One of the things you will discover early on is that you need some sort of XML editor. The language that makes the ribbon work is XML, but, in particular, a certain flavor of XML called **RibbonX**. There are different tools that will check your XML or RibbonX code for errors. Some of them may be purchased, some are free. I don't have a lot of money so I picked a free one called Bonfire. It works for me. You can Google it. I also tried a new razzmatazz version of XML Notepad that is supposed to do the job, but I didn't like it. It was too complicated. Ditto for VB.Net, even though I programmed in VB.net for over 3 years. Bonefire is adequate and free.

Using Bonfire you write your RibbonX code and save it to a file; an .xml file. The file, when saved, can be opened by any text editor, like Notepad (regular old Notepad). It is just a text file that ends with a .xml extension. Once the RibbonX code is written, you switch to Access and paste that RibbonX code into a memo field of row of an access table. You create the table yourself. Each row in the table will have a name and the memo field just mentioned. When you create a new form, or if you want to use your custom ribbon on a form you've already created, it is easily done. You simply move to the Ribbon Name property of the form and type in your ribbon name.

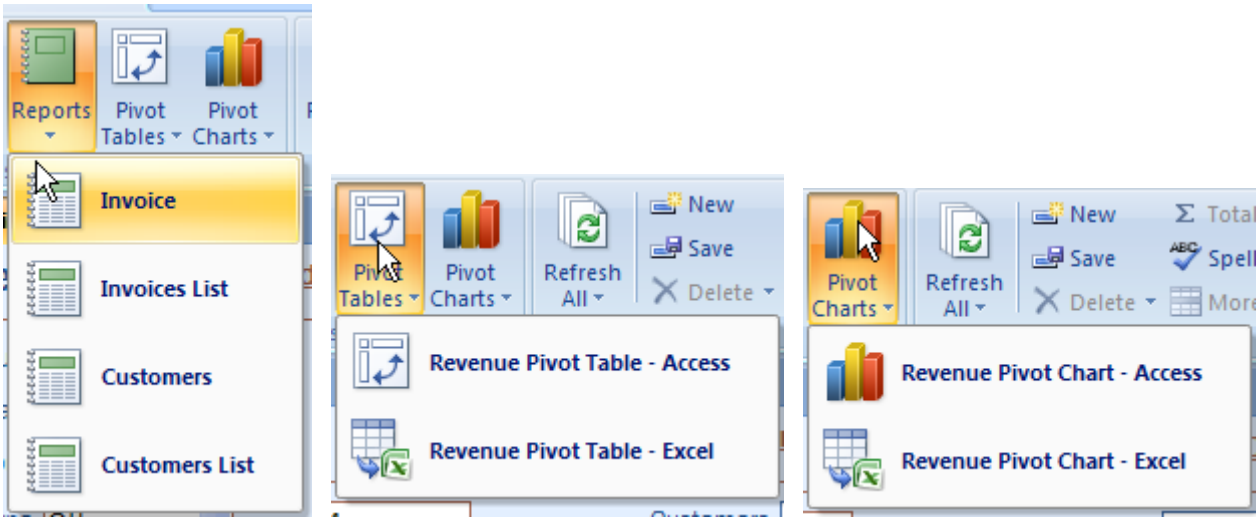
What your forms and reports really need:

Your forms and reports don't really need all that much by way of custom icons or ribbon groups; not if you're like me. Basically all I want the user to be able to do is open another form or report from some object on the ribbon. This is just like making a selection from a menu in 2003, only fancier.

So for a custom Form Ribbon you may want to have something like the following:

Now the above looks like it has a lot, and it does, but most of it is Microsoft supplied. Notice that there are really only three groups in this ribbon that are custom; the group on the far left called Forms/Reports/Views, and the two buttons on the far right called About and Exit Database.

The rest of the ribbon contains groups that are simply copies of standard Ribbon groups used elsewhere in Access. The two on the far right don't amount to anything. When they are clicked they simply open up another form. The buttons on the far left do a little more, but not much. These are not really buttons but menus, though they look like buttons. You can tell a menu from a button by the little down arrow associated with a menu. When the Reports menu is clicked a list of available reports is shown. Ditto for the other menu icons. Opened up, they look like this:



That's it. What could be simpler? As of right now, this form has most of the functionality of your average 2003 application and more. In fact, considerably more. You don't have to do anything to Sort or Filter or Find, Save, Refresh, Spell Check or Undo or ReDo. All of that is built in. How cool is that? It begins to make Ribbons seem more pleasant for the developer, as well as the user.

Fancier Things:

From this point on I hope the reader will have before him/her, the associated .xml file that goes with this white paper. If you look at it and the Ribbon above, the rest of what I say will make sense. Without it, it may not.

If you want, you can do things that are fancier that what I have done above, but you don't have to. It's just not necessary. Anything and everything that Microsoft does with their ribbons is available to you, and if you do it all the Microsoft way, it will get complex. But the user doesn't care. If you present the user with a menu (like above) or a dynamic combo box when picking a report, do you think he/she really cares? All the user wants to do is get things done quickly and easily. This you can provide, without getting needlessly complex.

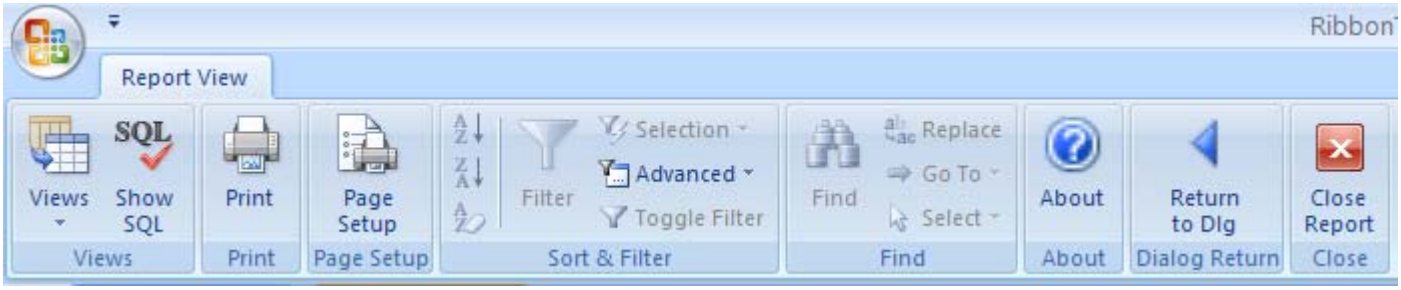
The XML:

If all you had to do was study XML life would not be difficult. XML is just another language or way of doing things that is unique to itself. Personally, I don't want to make a career out of writing XML. But Microsoft has decided to use it as the tool for linking things together so I have to know enough (just enough) to play the game. If you stop to think about it, XML is somewhat analogous to a transmission in a car. With a car you have an engine, a driver, and a compartment for the driver to sit in. Great! So far so good. So now, how does the driver make the car do something? Well, he moves the little knob between him and the passenger to "Drive" and drives off. If he wants to go backwards, he moves the knob to reverse.

With a ribbon you provide an interface to the transmission (a knob or lever that allows you to select Drive or Reverse or Park).

Writing the XML is the most intimidating part of making RibbonX work but it's not so hard with Copy and Paste. That is, all you have to do is find something that somebody else has written that does approximately what you want, copy and paste it into your.xml editor, and modify it to meet your own needs. Not only that but once this is done, once the problem is solved one time, you can copy and past and use if over and over again for a myriad of solutions to other problems.

As mentioned, there are two .XML files that goes along with this document. One of them contains the XML code for a form (see form ribbon above) and the other contains XML for a report. The report ribbon looks like this.



I would include the XML for these ribbons hear, in this document, but I am writing this in WORD and XML does not format well in a WORD environment.

The long and short of it for the Form Ribbon:

I will talk here about the form ribbon, but most of what I say applies to the Report Ribbon as well. In order to produce the Form Ribbon you're going to need to write some RibbonX code in an XML editor. At the top of the XML code (refer to the accompanying .xml file) you're going to include a line that tells Access to start from scratch with regard to your new ribbon. In other words, when this file is opened by Access the first thing it's going to tell Access is "take any ribbon logic you are already be aware of and forget it. I'm going to start from scratch."

Next, create a tab called "Solar with a Flare". You can have more than one tab if you like, but for me, it is easiest to create one tab and then put all the groups I want inside that tab. Some of the groups will be custom and created by me, others will be copies of groups Microsoft already has made.

When you study the code and look at the ribbon you'll see that I created a custom group and gave it a name and a label. The label is "Forms/Reports/Views". Then, I identified the objects that I want included in that group; the menus that look like buttons.

When the menu is clicked it reveals real buttons. When the user selects one of the buttons something happens; like a report is run or a form is opened. Each line in the XML that refers to a button requires that you supply it with information. You need to supply it with a button ID (a name of your choosing), a button label (like "Invoice List") an Image Name (what kind of button is it going to be--How's it going to look) and (and this may be the most important one) the name of the Public Sub routine you want called when the item is selected. If the Sub is public and in a module somewhere, it will be found automatically by Access when the item is clicked and the code in the sub will run.

One other thing you may need to supply is the Tag. With the tag line, you get more specific about what you want the sub to do when it is called. For example: In the line mentioned above, the XML calls a sub routine called **OnClickReportsMenu**. This resides in a module named **modRibbonCallBacks**. I want this sub routine called each time a button (any button) in the Reports menu is selected, so, no matter what report you select, this sub routine should be called. I do it this way because, regardless of the report that is called, I want a certain thing done first. I want to run a routine that posts changes to the form before calling any report. By having each menu item button call the same sub, this solves that problem.

But, now that this sub has been called, we need to identify which button actually called it. This is what the tag line is for. With those two pieces of information we can write VBA code to get the application to do anything we want. I am not real big on trying to explain every line of code to others. The problem is not with you; It's with me. I don't have the patience. If I make the smallest mistake it become more confusing than helpful. So, I won't go into a lot of detail here about the XML or VBA code. For the XML, I'll let the books (already mentioned) do that. For the VBA, I will just include a snippet below and give some minimal explanation.

```
Public Sub onClickReportsMenu(ctrl As IRibbonControl)

    If Not CheckAndSaveDirty Then Exit Sub

    If ctrl.Tag = "Invoice" Then
        DoCmd.OpenForm "frm002_InvRpt_dlg"
    ElseIf ctrl.Tag = "Invoices List" Then…
```

As you can see, in the code above the first thing that happens is that the sub calls another sub routine called CheckAndSaveDirty(). This is a sub I've written that checks to see if the form is dirty and if so, it posts changes before allowing the report to be called. Then, it checks to see which button was actually selected from the reports menu. This is identified by the tag line. If the tag = "Invoice", then I know the user wants to print the "Invoice" report. Some reports requires that a dialog be opened first so the application can get criteria information from the user. DoCmd opens this form and the control of the logic is passed off to it. Then, of course, there is an ElseIf statement that checks the tag for all the other possible items (reports) the user may have picked. That's it. It's really no big deal.

The one thing I didn't cover is the **argument** that is passed to the **onClickReportsMenu** method of the **modRibbonCallBacks** module. . This is an area that is still a little fuzzy to me but I will make an attempt to explain it below under **The Associated Database**. But my lack of understanding goes to illustrate a point. It's always better when you know what you're doing, but I don't have to know the theory of electricity to turn on a light switch. All of the things I am discussing here are covered in greater detail in

the Access 2007 Inside and Out book. See particularly page 1297 of the Access Inside and Out book under the heading of Dynamically Updating Ribbon Elements.

The Summary:

I realize I have not taken a step by step approach here. But the books do. And if you print out the associated xml file, study the custom ribbon, and read this text, I think it will all make sense.

Basically, all I care about is giving the user a pleasant experience when using my Access application, and also, staying in the technology main stream. There may be some question today as to whether or not Access is still truly mainstream. I don't see how it can be considered anything but mainstream given that

1). It is the most ubiquitous database in the world, and
2). It is a Microsoft product.

To my mind it's not a question of whether Microsoft is in the mainstream, the question is whether Access developers will stay up with Microsoft and switch to the newer way of doing things. Personally, I think that if you limit yourself to a version of Access that's six years old you've taken yourself out of the mainstream.

If there are others that want to go the .Net route I say God bless you. I have great respect for my brethren who program in this tool. But remember that 80% of the reason for creating .Net (maybe it was 90%) had to do with the Internet. It is primarily an Internet tool, that also happens to do client server. The real question may be, is Client/Server dead? Again, I think not. I know it's not in favor, at present, but that's because there's a whole generation of people out there who don't know the benefits of client/server as opposed to the web. That's a whole different white paper, but the bottom line is this; at least, this is how I look at it and how I advise potential clientele; You never want to write a web application instead of a client/server application, unless there is a clear need. Why? Because for one thing it will prove to be twice as expensive and complicated.

And, if you *are* going to write a client/server application, you never want to do it in .NET instead of Access, again, unless there is a very clear need. Why? Same reason. It is going to be much more expensive and much more complex, and as a result, harder for the client to maintain. If that complexity is truly needed, then let me be the first to advise you to do it that way. But much of the time, the expense and complexity is completely unnecessary. Whenever possible, simple is always better.

You can basically run a small to medium size business with a good client/server application written in Access. And the best part for the client is this: If you don't like your programmer/developer, they are easily replaced. Plug and Play is the key phrase here. Unplug one, plug in another, If the client is unhappy with the programmer/developer they will always be able to find a replacement and one that, at a max, will bill out at half the rate of a good dot NET developer. I say to the client, "insist on ownership of the application and orthodoxy in code and there should be no problems." You (the client) will be in control, and you will not be in a position where an individual, good intentions or not, has got you snookered, or worse yet, on your knees.

The Associated Database:

Explanation:

The database that goes with this white paper was written from scratch. It has very little in it; a couple of tables, a couple of forms, a report and a module called **modRibbonCallBacks**. This module is key to making custom ribbons work. When I created the database I copied this module over from another

database and just made a few modifications to make it adapt. To see how ribbons work, run frm001_Invoices. Here is the OverView of what's happening here, as best I understand it.

When a frm001_Invoices is opened, it checks to see if there is a value in the Ribbon name property for the form. If there is, that ribbon will have a name like rbnRibbonDemo_01. The code (the xml) for rbnRibbonDemo_01 is kept in a system table of the database called USysRibbons. This is a table we created. We wrote our XML for our Ribbons in an XML editor, like Bonfire, but we then copied that code to a memo field of a particular row of the USysRibbons table. When the form opens, it fetches this code and stores it in memory.

The code in the XML needs some way to communicate with the VBA you write. That is, when a button or menu item on the ribbon is clicked, the XML points to certain public sub routines in VBA. These sub routines are kept in the modRibbonCallbacks module.

How to explain all this is not entirely clear to me. I suppose if I were a purest and I felt my manhood was involved, I'd make a greater effort to understand and explain this, but such is not the case. Suffice it to say that when opening a form Access knows that it needs to run the OnRibbonLoad1 sub routine of the modRibbonsCallback module. It gets this from the first line of the XML

In the XML you have code that says, "When I press this button, run a sub routine in modRibbonCallback called "X". For example, in the XML for the form, when the user selects an item in the reports menu, the code in the XML says, run a sub routine called OnClickReportsMenu in the Access module called modRibbonCallback. The onClickReportMenu sub looks like this:

```
Public Sub onClickReportsMenu(ctrl As IRibbonControl)
    'Place your code here
End Sub
```

With all of the above in place things just work. It's just like 2003 except that, the menus are a fancier, and the setting up of the Ribbon was more complicated. More complicated, but once you figure out the above, it can be used and re-used over and over again, and as a result, you can make the ribbon do anything you used to do with plain ole menus.

Additionally:

There are several functions and sub routines in the modRibbonCallBacks module, but really only the first two items are of interest here, all located at the top of the module. The global variable called gobjRibbon1 is important, as are onRibbonLoad1() and onClickReportsMenu(). The rest of the code has to do with things that are necessary to make the form work but not really relative to the problem at hand.

The point to remember is how easy it is to get this all working. With just a global variable and a couple of sub routines, you're in business.

Other Odds and Ends:

I mentioned earlier that RibbonX can get quite complicated, and it's true. Most of the complexity is unnecessary, most of the time, but if you do need to do something fancy, there is a very good book out called **RibbonX: Customizing the Office 2007 Ribbon**. It is 700 pages and goes into every detail. The authors are Robert Martin, Ken Puls and Teresa Henning.

Additionally:

There are third party tools that make creating custom ribbons "point and click" easy. For me, they have their limitations and I prefer not to use them because what I've written above is not that hard to do or implement. I personally don't like to give a third party tool that much power over what I do unless there's no other choice. But, if you feel comfortable with it, why not.

Also:

My guess is (though it is just a guess) Microsoft itself will do something in the next version of Access to make ribbons a little easier to deal with. It would certainly be in their best interest.

Author:

Woody Splawn – Owner Information Solutions – Roseville CA (a suburb of Sacramento)
Email: WSplawn@Surewest.Net

Phones: (916) 782-5393 (916) 316-9032 (cell)
WEB Site: http://microsoftaccesssacramento.weebly.com/