Outputting Flexible Data

Dave Gannon and Nich Mann

Producing static reports is easy—but what if your users want to be able to customize their output? Dave Gannon and Nich Mann look at all the options available to you and let you in on the best answer.

We were recently posed a problem by a client: The user wanted to produce, from Access, a set of letters to go to customers (see Figure 1 for a copy of the sample letter). Each letter included a number of line-item records (a different number for every customer) in the body of the document. The problem was that the user needed to be able to customize the format of the letter at any time—and without calling us.

We decided that the best solution was to use Word in some way, allowing the user to modify the format of the document in Word. Most developers think of paper-based output coming directly from Access, but using Word to produce output documents for Access data is often a great way to go, especially if the end users value flexibility in their output. From Access, we'd create new copies of the letter and insert our data.

Having selected Word as our document generator, we realized that there were various ways that we could use Word. In this article we'll show how we solved the problem and what solution we picked. Before that, though, we need to provide some background on ways of outputting data from Access in printed format. There are three ways of getting data into a Word document:

- OutputTo method
- Mail-merge
- Automation

Each has its advantages and disadvantages.

Figure 1. Our original document, shown as an Access report.

OutputTo method

Using OutputTo is probably the simplest way of exporting your reports to Word. The effect is the same as when you click the "Publish It with Microsoft Word" button (see Figure 2).

It only takes one line of code to output a report in this way, so the primary benefit is that this is a simple way of exporting data:

```
DoCmd.OutputTo acOutputReport, "rptClientOrders", _
     acFormatRTF, "MyReport.RTF", True, "Report.DOT"
```

This line of code will prompt the user for where to save a copy of an Access report. The report will be exported as a rich text format document. The output document will be called MyReport.RTF. The True following the document name will cause Word to be started immediately after the document is created and display the document to the user. The last parameter in the call specifies a Word template document that will

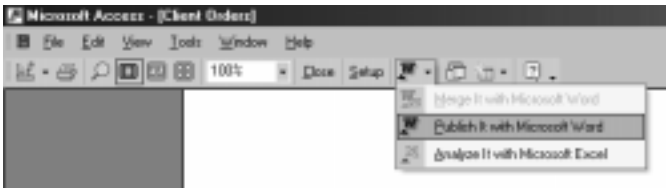

Figure 2. Outputting data to Microsoft Word.

control the appearance of the RTF document.

Although there's virtually no code to support with this method, this simplicity comes at a price. The result is rich text (rather than a native Word document), so a lot of formatting is lost: Graphics don't get exported, including any lines/boxes/circles. Figure 3 shows the result. In the sample database in this month's Download, we've included our sample report and the code to output the report as an RTF document. In our case, these graphic components were essential items to include in the end result. Because the end result was a letter that would get sent to external customers, the letter needed to look as good as possible.

Automation

Automation is a great way of outputting information when you need to have a high level of control over the output. Anything you can do in Word, you can program Access to tell Word to do. It's easiest to automate Word when you know what version you have to automate, allowing you to specify the version of Word to automate in your code. This early binding (specifying the version of Word when writing your code) gives you a performance advantage, plus you get IntelliSense support when you're programming. If you have this luxury, it's easiest to use early binding.

As luck would have it, though, our client was in the middle of upgrading their office systems: Users would be

Order

Lenny

Thank you for placing an order with us. Please find below the items you ordered. If for any reason you need to contact us to discuss this order, get in touch with us on +44 (0)113 5551234.

Order Reference 0000007

Item Name	Cost	Qty	Total Cost
Big Wheel	£75.00	1	$75.00
Huge	£100.0	2	$200.0
Medium	£50.00	2	$100.0
Small	£30.00	5	$150.0
Tiny Wheel	£20.00	8	$160.0
Grant Total		18	6685.00

Many thanks for choosing McFly's Wheels

Figure 3. The report in RTF format.

using different versions of Word, even though they'd all be using the same version of Access! The most sensible approach then would have been to use late binding: opening Word without specifying the specific version being used.

When you're using late binding, your code looks like this:

```
Dim wordObj As Object 'Word.Application
Set wordObj = CreateObject("Word.Application")

With wordObj
     Visible = True
    .WindowState = 1 'constant for wdWindowStateMaximize
    .Documents.Add Template:= _
       "C:\TemplateDirectory\WordTemplate.dot", _
       NewTemplate:=False
     If .ActiveDocument.Bookmarks.exists("FullName") _
                                        = True Then
        .Selection.Goto What:=-1, Name:="FullName"
        .Selection.TypeText "Jason Patrick"
     End If
End With
```

Late binding is slower than early binding. You also don't have the luxury of using Word's built-in constants, so you'll find yourself getting familiar with the object browser in Word to determine the numeric values of constants (for instance, wdWindowStateMaximize).

When you control Word this way, you have unparalleled control over Word, and can produce dramatic looking reports. This power comes at a price, though. Even with early binding, it's unbelievably slow. This isn't a problem when you're preparing a bill of materials for a single client, but it's interminable when generating documents for a few thousand clients: Start it running, then go home. It might be ready for you tomorrow. If you're lucky.

Clearly, this was unacceptable for a client who wanted to produce a print run today.

Mail-merge

Most Word users are familiar with mail-merge. Mail-merge allows a Word document to be used as a template that's linked to a data source—such as Excel or Access. The template is then used to create a separate document that contains the data source's information merged with the content in the template. The resulting document is generated on a page-per-record basis.

The advantages are easy to see. The user can format any text content. The user can change the document pre-merge or post-merge. The location of the data—where it appears in the final document—is easily set up and altered. You can even automate the mail-merge process from within Access (although this is beyond the scope of this article).

The disadvantage is the "page-per-record" restriction since we needed to put multiple records in each letter.

The solution

Now that we've described each method, you may be able

to answer our original problem. Access reports aren't feasible because the client needs to customize the letter. Automation could work, but if the client changed the letter drastically it could cause our code to fail. So, mail-merge seemed to be the answer: The client could customize the document, even relocating where the merge fields were to be placed (including adding and removing data fields). But how to put many records on one page?

The answer was to accept that Word expects one record, and use one record per customer. However, the record that we inserted into the document had the individual line items in the record, each record separated by carriage returns in the individual fields. As an example, assume that the following represents a typical query result:

John Smith	Doughnuts	£2.50
John Smith	Strawberry Jam	£1.26
John Smith	White bread	£0.89
John Smith	Butter	£0.97
Tony Greaves	Olive Oil	£7.50
Tony Greaves	Strawberry Jam	£1.26
Tony Greaves	Coffee	£2.65

If you used this query as the basis of a mail-merge form letter in Word, you'd end up with a seven-page report (a line item for each page). Concatenating each line item into a single record per customer, you'd end up with this output:

John Smith	Doughnuts	£2.50
	Strawberry Jam	£1.26
	White bread	£0.8
	Butter	£0.97
Tony Greaves	Olive Oil	£7.50
	Strawberry Jam	£1.26
	Coffee	£2.65

This query can now successfully be used as the basis of a mail-merge to list all items of a customer on one letter. We felt that we had an elegant and simple solution.

Well, actually, it wasn't so simple. The difficulty lies in the fact that SQL is set-based. This means that there's no simple SQL query that will give you this result. Because each line item is stored as a separate record, SQL will always return a single record per line item. For this solution to work, then, we needed to dip into some code.

Analyzing the problem in pseudo-code would give this:

```
For each person
 Concatenate all line item descriptions
 Concatenate all line item prices
Next person
```

First, though, we needed a query that gave all the source data for the customer (shown in Figure 4). We need the customer id (the field [ClientRef]), the item cost, and the item quantity. The query was sorted on the customer id in ascending order.

Now all that we needed was a routine to concatenate the results of the query into a table that could be used as a data source for our Word mail-merge document. The code begins with some variables:

```
Dim adoConn As New ADODB.Connection
Dim adoCmd As New ADODB.Command
Dim adoRs As ADODB.Recordset
Dim adoOutputRs As New ADODB.Recordset
```

These are fairly obvious, but we also needed some variables to store data that will let us know when the customer id changed and to store the orders and costs for each customer:

```
Dim lCustRef As Long
Dim lLastCustRef As Long
Dim sOrders As String
Dim sCosts As String
```

We began by opening the connection and the recordsets:

```
With adoConn
  .ConnectionString = CurrentProject.Connection
  .Open
End With

With adoCmd
  .ActiveConnection = adoConn
  .CommandType = adCmdStoredProc
  .CommandText = "qryOrderItems"
End With

Set adoRs = adoCmd.Execute
```

The next steps are to create the output recordset, and initialize the temp holding variables:

```
adoOutputRs.Open _
  "SELECT * FROM tblConcatenatedOrders", _
  adoConn, adOpenKeyset, adLockOptimistic

lCustRef = adoRs!ClientRef
lLastCustRef = lCustRef
sOrders = ""
sCosts = ""
```

We then looped through the source recordset, copying the line items into the temporary variables and following that each line item with a carriage return and a line feed

Continues on page 18

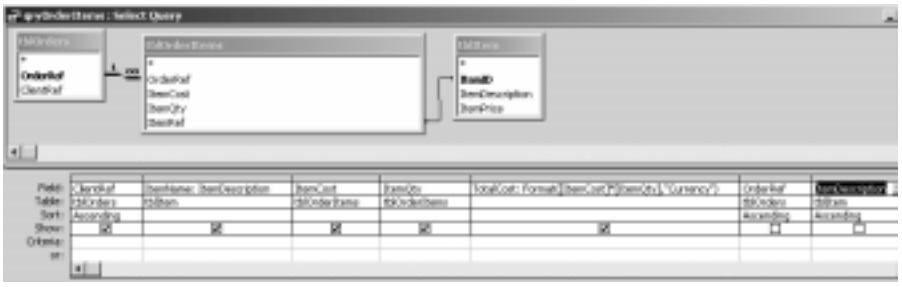

Figure 4. The base query for retrieving customer data.

Outputting Flexible Data...

Continued from page 15

to start a new line. When the customer id changes, we added a row to the target recordset and continued:

```
Do While Not adoRs.EOF
  With adoRs
    lCustRef = !ClientRef
    sOrders = sOrders & !ItemName & vbCrLf
    sCosts = sCosts & !TotalCost & vbCrLf If
    lLastCustRef <> lCustRef Then

     adoOutputRs.AddNew
     adoOutputRs!CustomerID = lLastCustRef
     adoOutputRs!Orders = sOrders
     adoOutputRs!Costs = sCosts
     adoOutputRs.Update
     sOrders =
     "" sCosts =
     ""
     lLastCustRef = lCustRef
   End If
   .MoveNext
 End With
Loop
```

And that's it. This simplified version of ourtenation routine doesn't clear out the target table before it starts and has no error handling. We leave those additions to your best judgment. The sample database contains a more complete recordset than the simple one used here.

We should also point out that there are disadvantages to using this method. Performance could be an issue, depending upon the number of records retrievedand the number of items per customer. The other disadvantage is that really this method is intended for small, uncomplicated queries. A query with many fields that needed to be concatenated could prove to be too code-intensive, and you'd need to alter the code every time the user wanted a new field added to the letter.

This method is neither revolutionary nor immensely complicated. We've worked with Access for many years and spotted this method only after the question was posed. In many ways, the main point of this article is to emphasize that you should always try to keep an open mind and try new ways of using familiar techniques and knowledge. Only by doing this can you open up possibilities of creating applications that are more interesting both for you and your users. ▲

WORDREPT.ZIP at www.vb123.com/kb

Dave Gannon and Nich Mann have more than 15 years of Access development experience between them. They currently work in Harrogate, England, producing bespoke sharescheme administration solutions for Howells Associates Ltd