Excel Macro Combines Financial and Non-Financial Data

By Dan Hill, CMA*

An unavoidable fact of life is that data must be gathered before analysis can begin. Banking and other industries measure effectiveness using financial data, found on the General Ledger (GL) system, and non-financial data, found on core application systems. Data must be collected from different computer systems, and therein lies the problem. How to blend the data together fast? One way to combine financial and non-financial data is to use Microsoft's Visual Basic for Applications (VBA), a.k.a. Excel macros.

To demonstrate how easily Excel macros can manage such an analytical challenge, this article examines a macro that combines financial and non-financial data for each item in a list. Here a list of bank branches is used, but with simple modifications the same macro will cycle through any type of list: customers, contacts, products, regions, departments, lots, assets, dates, file-names, etc., etc. The many purposes to which you can apply this macro are limited only by your needs and imagination.

FOUR BASIC STEPS

The macro has four basic steps:

(1) Startup chores,

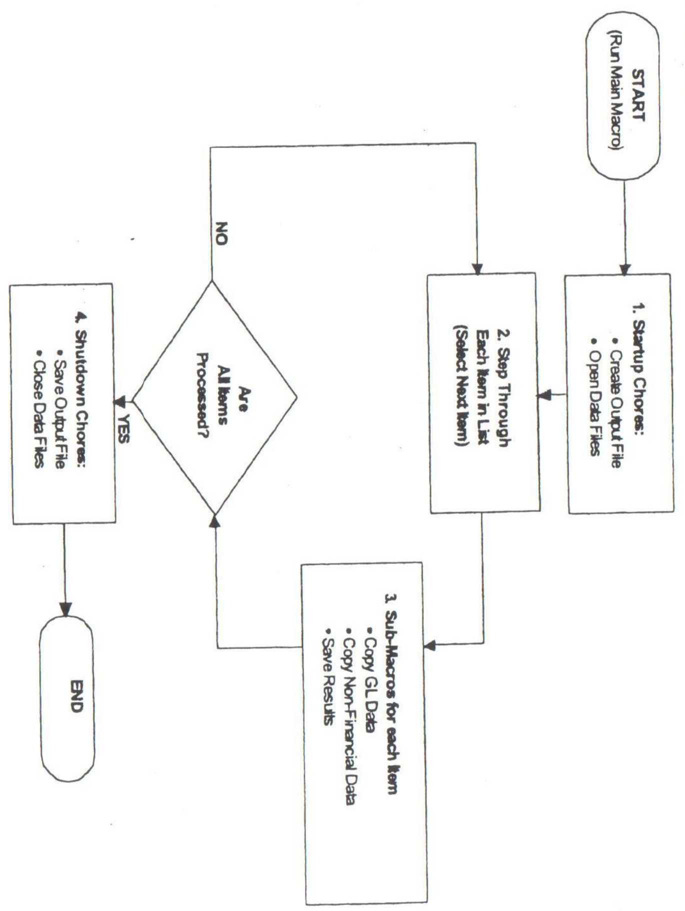
(2) Selecting each item in a list, one item at a time,

(3) For each item performing one or more operations, and

(4) Shutdown chores (See Figure 1).

*Vice President, Bank of America, dnhill@mindspring.com. Mr. Hill has an MBA fron George Mason University.

FIGURE 1
Macro has four basic steps



Macro automation does only part of the job. The macro retrieves data for each branch and saves the results, but all calculations are performed using normal Excel formulas. This way data is pulled into the familiar, and powerful, spreadsheet environment. Further, the list of items — in this case bank branches — is easily maintained in its own worksheet.

Most importantly, data files are accessed in whatever structure they happen to arrive. ASCII or text files received from mainframe systems must be converted to Excel workbooks, but the structures of these files do not need to be altered.

First Things First

Before coding the VBA language you must attend to some preliminary tasks. For starters prepare the list of items — or branches — that the macro will cycle through. Branches are represented by cost center numbers, which are maintained in the sheet CC_List (see Table 1). The CC_List sheet also contains each cost center's branch name and save sheet name.

The next preparatory task is setting up the main workbook, which receives the disparate data and performs the calculations. I call the main workbook MasterFile and give it a filename of MyMacro.xls. A GL_Data sheet is created in the main workbook to receive and display the imported GL data (see Tables 2 and 3).

Also included in the main workbook is the Calculator sheet (see Table 4). The Calculator sheet is where your hard work pays off. It is here that you will reference imported data, perform calculations, and display powerful relationships.

Finally, consider the structure of the external data files. The simplest structure is to list fieldnames across the columns and records (e.g., cost centers) down the rows, a flat table. (Table 3 is an example of a flat table). The macro described here retrieves data from just such a structure. Data can come in other structures, of course, and with modifications the macro can handle most any structure.

Your preparatory tasks are done. Now it's time to code the VBA language and make the whole thing work!

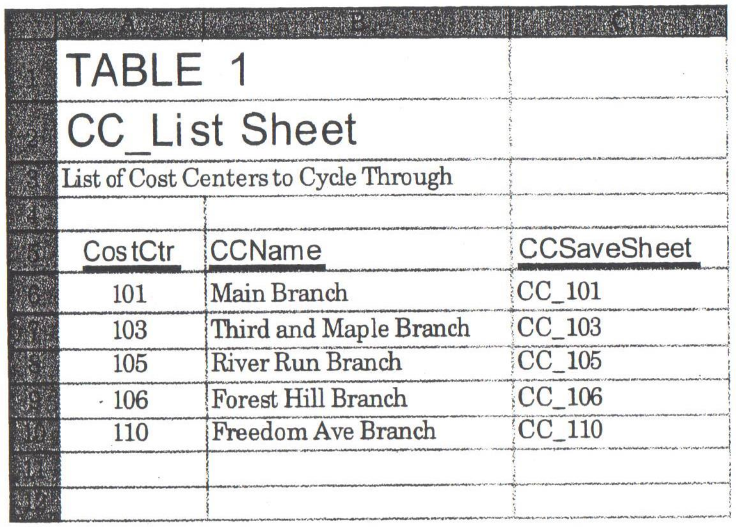


TABLE 1

CC_List Sheet

List of Cost Centers to Cycle Through

CostCtr	CCName	CCSaveSheet
101	Main Branch	CC_101
103	Third and Maple Branch	CC_103
105	River Run Branch	CC_105
106	Forest Hill Branch	CC_106
110	Freedom Ave Branch	CC_110

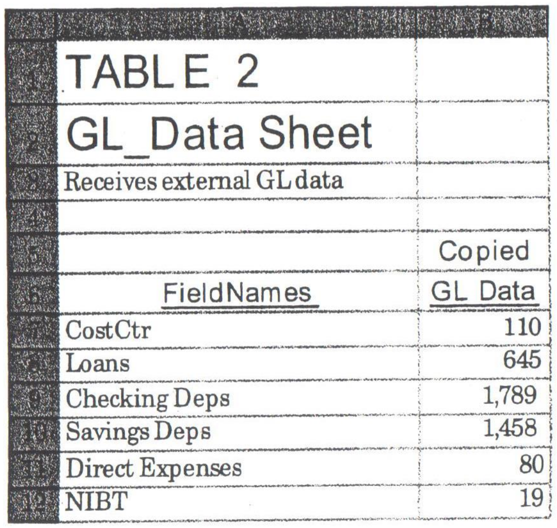


TABLE 2

GL_Data Sheet

Receives external GL data

	Copied
FieldNames	GL Data
CostCtr	110
Loans	645
Checking Deps	1,789
Savings Deps	1,458
Direct Expenses	80
NIBT	19

TABLE 3

GL Data File: GLData.xls

External GL data file (a flat table structure)

CostCtr	Loans	Checking Deps	Savings Deps	Direct Expenses	NIBT
101	2,455	10,313	5,120	525	125
103	6,450	17,890	14,580	795	190
105	8,708	24,152	19,683	1,073	257
106	3,683	15,470	7,680	788	188
110	645	1,789	1,458	80	19

TABLE 4

Calculator Sheet

References imported data and makes calculations

Branch Cost Center	110
Total Loans	$ 645
Total Deposits	$ 3,247
Average Balance per Acct: Checking	$ 983
Average Balance per Acct: Savings	$ 919
Direct Expenses per Acct	$ 23.34
NIBT per Account	$ 5.58

FLAT TABLE LAYOUT

CstCtr	Tot Loans	Tot Deps	BalPerAcct Ckg	BalPerAcct Sav	ExpPerAcct	NIBTperAcct
110	645	3,247	983	919	23.34	5.58

Let's Write a Macro

You'll type the Visual Basic code into the main workbook — this way the code is saved and retrieved with the main workbook. The steps taken to open the Visual Basic editor depend on your version of Excel.

For Excel version 97: Open the main workbook file. Choose Tools, Macro, Visual Basic Editor. Once inside the VBA editor, choose Insert from the toolbar, then Module. The VBA module contains the VBA editor, which is designed to handle the Visual Basic code.

For Excel version 95 or earlier: Open the main workbook file. Choose Insert, Macro, Module. The Module sheet contains the VBA editor, which is designed to handle the Visual Basic code.

For Everybody: When writing VBA code use an apostrophe ' to make comments. Comments placed throughout your code will make it much easier to review and debug. Ditto for indenting. The VBA program ignores white spaces, so why not indent, indent, indent!

Below describes the VBA code in Figure 2 (see page 46).

FIGURE 2: VBA CODE

```
' MyMacro July 2000: Copies data by Cost Center from several data sources
' Declare Constants
    Option Explicit
    Private Const MasterFile = "MyMacro.xls"         'The main workbook
    Private Const DataDir = "D:\macros\"             'Dir for input and output
    Private Const GLFile = "GLData.xls"              'GL Data file
    Private Const SaveFile = "Results.xls"           'File to save results

' Declare Variables
    Dim CostCtr As String                            'Cost center being processed
    Dim CCSaveSheet As String                        'Sheet name to save CostCtr's results

' Do_It_All Macro: The main macro
Sub Do_It_All()
    Workbooks.Add                                    'Create the output file
    ActiveWorkbook.SaveAs filename:=(DataDir & SaveFile)

    Workbooks.Open filename:=(DataDir & GLFile)      'Open data file

    Windows(MasterFile).Activate                     'Go to cost center list
    Sheets("CC_List").Select
```

continued on next page

```
    Cells(6, 1).Select                               'Select the 1st cost center

    While ActiveCell <> ""                           'While selected cell is not blank
      CostCtr = ActiveCell.Text                      'Set cost center variable
      CCSaveSheet = ActiveCell.Offset(0, 2).Text        'Set sheetname variable

      Copy_GLData                              'Sub-macro: Copy GL Data for selected cost ctr
      Save_Results                             'Sub-macro: Save results for selected cost ctr

      Windows(MasterFile).Activate                   'Go to cost center list
      Sheets("CC_List").Select
      ActiveCell.Offset(1, 0).Select                 'Move down one row to next cost center

    Wend                                             'Repeat while loop for next cost center

    Windows(SaveFile).Activate                       'Save the output file
    ActiveWorkbook.SaveAs filename:=(DataDir & SaveFile)
    ActiveWorkbook.Close                             'Close the output file

    Windows(GLFile).Activate                         'Close the data file
    ActiveWorkbook.Close

End Sub                                              'The main macro comes to an end

' Copy_GLData Sub-Macro: Copies GL Data for selected cost center
Sub Copy_GLData()
    Windows(GLFile).Activate                         'Go to GL data file
    Sheets("Sheet1").Select
    Cells(6, 1).Select                               'Select first cost center in file

    Do                                               'Initiate a do loop
      If ActiveCell = "" Then                        'If selected cell is blank (EOF)
        End                                          'End execution: CostCtr has no data
      ElseIf ActiveCell <> CostCtr Then              'Else If selected cell not equal
        ActiveCell.Offset(1, 0).Select               'to CostCtr variable, then move
      End If                                         'down one row to next cost ctr

    Loop Until ActiveCell = CostCtr            'Loop until selected cell = CostCtr variable

    'Select for copying GL data for selected CostCtr
    Range(ActiveCell, ActiveCell.Offset(0, 5)).Select
    Selection.Copy

    Windows(MasterFile).Activate                     'Copy the GL data
      Sheets("GL_Data").Select
      Range("B7").Select
      Selection.PasteSpecial Paste:=xlValues, Transpose:=True

End Sub                                        'Return control to Do_It_All main macro
```

continued on next page

```
' Save_Results Sub-Macro: Saves selected CostCtr's results to Output file
Sub Save_Results()
   Windows(SaveFile).Activate                'Go to the Output file
   ActiveWorkbook.Sheets.Add                 'Create a new sheet
   ActiveSheet.Name = CCSaveSheet            'Name sheet CostCtr's sheetname

   Windows(MasterFile).Activate              'Go to the Calculator sheet
     Sheets("Calculator").Select
     Calculate                               'Calculate before copying
     Cells.Select                            'Mark to copy all cells
     Selection.Copy

   Windows(SaveFile).Activate                'Go to the Output file
   Sheets(CCSaveSheet).Select                'Go to selected CostCtr's sheet
     Cells.Select                            'Select all cells
     Selection.PasteSpecial Paste:=xlValues  'Paste values
     Selection.PasteSpecial Paste:=xlFormats 'Paste formats

End Sub                                  'Return control to Do_It_All main macro
```

Much of what you type when coding must be typed exactly as shown. These are code keywords or operators, required by the VBA language. But the code writer — that's you — makes up nicknames for variables and macro programs. I'll use **BOLD TYPE** to indicate made-up nicknames. You do not need to bold made-up names when you're writing code. I'm bolding here so you can distinguish between required syntax and made-up nicknames.

The first line of code gives a title and a brief description.

'MyMacro July 2000: Copies data by Cost Center

Notice the apostrophe, which tells the macro to ignore the rest of the line; it's for human consumption only!

Next you declare the constants. Constants are values that won't change as the macro runs. For example, it's a good idea to declare as constants file names and directories. This way you won't bury file names and directories deep inside the VBA code.

Option Explicit
Private Const **MasterFile** = "MyMacro.xls"
Private Const **DataDir** = "D:\macros\"

The constants assigned are nicknamed MasterFile and DataDir. MasterFile is equal to an Excel filename, and DataDir is equal to a directory.

The Option Explicit statement forces you to declare constants or variables (discussed below) before using them in the macro. Believe it or not, this is a big help. It keeps you from accidentally misspelling a constant or variable name.

Next you declare — and nickname — each variable.

Dim **CostCtr** As String

Variables are temporary values that change as the macro runs. For example, the cost center currently being processed is a variable, the CostCtr variable. The value of the CostCtr variable will change once the current loop is completed and it's time to move on to the next cost center.

The Do-It-All

Now it's time to code the main macro. I call it the Do_It_All macro. The main macro is the big boss that controls the whole show. It does the housekeeping and calls up sub-macros to do specific jobs. Use a Sub statement to start the macro — or sub-procedure — nicknamed the Do_It_All.

Sub **Do_It_All**()

The first order of business is the start-up chores, which include creating the SaveFile workbook to receive the results.

Workbooks.Add
ActiveWorkbook.SaveAs filename:=(**DataDir** & **SaveFile**)

The new workbook is saved to establish its file name for reference later in the macro. The ampersand & symbol combines the constants DataDir and SaveFile to create a complete path and filename for Excel to use: D:\macros\results.xls.

Another start-up chore is opening the data files so they'll be available for use.

Workbooks.Open filename:=(**DataDir** & **GLFile**)

Let's Go Stepping

The startup chores are done, and the main macro can now begin stepping through the list of items — in this case the cost center list. The macro activates the main workbook window and selects the CC_List sheet (Table 1).

```
Windows(MasterFile).Activate
Sheets("CC_List").Select
```

You Activate a window but Select a sheet. Notice the quotation marks around the sheet name "CC_List". Whatever is inside quotes is interpreted exactly as typed; in this case the tab name of the sheet you're selecting. On the other hand MasterFile is not surrounded by quotes, which tells the macro that MasterFile is a variable or a constant and requires a value.

Now select the cell containing the first cost center in the list. There are different ways to select a cell. For example, the results of the following two lines are identical:

```
Range("A6").Select       'Select cell A6
Cells(6, 1).Select       'Select row 6, column 1 (cell A6)
```

The Cells nomenclature is in row by column order. Using row by column numbers has its advantages, as you will soon see.

Next comes a While loop, which is used to move down the list of cost centers one at a time.

```
While ActiveCell<> ""
```

The object ActiveCell is VBA code for the selected cell. The above says while the ActiveCell is not<>blank "", perform the following operations. You want the While loop to stop when a blank cell is reached, that is, when you reach the end of the cost center list.

So, as long as the selected cell is not blank, then it contains a cost center to manipulate. Assign the cost center number in the selected cell (the ActiveCell) to the variable CostCtr.

```
CostCtr = ActiveCell.Text
```

For example, if the selected cell is A6 on the CC_List sheet, then CostCtr is assigned the value 101 (see Table 1).

Assign variables that are related to the current cost center using the offset method.

```
CCSaveSheet = ActiveCell.Offset(0, 2).Text
```

With the Offset method you offset or count, relative to the current position, the numbers of rows and columns to move. In this case you're starting at the cell containing the selected cost center (the ActiveCell) and then offsetting no rows down and two columns to the right. If the selected cell is A6 on the CC_List sheet, then CCSaveSheet is assigned the value CC_101 (see Table 1).

Now that a cost center has been selected and the CostCtr variable assigned, it's time to call up the sub-macros to do specific jobs for the current CostCtr.

```
Copy_GLData
Save_Results
```

It's a good idea to put your more complicated routines into separate sub-macros. This helps in designing and debugging your code. It also allows you to "comment out" a sub-macro so it won't run. You do this by putting a temporary apostrophe in front of the sub-macro's name in the Do_It_All main macro. This is a major convenience when building and testing other sub-macros.

Finish the Do-It-All

I'm going to postpone discussing the sub-macros for a moment. For now, let's finish going through the Do_It_All main macro.

Once all the sub-macros have run for the current cost center, then go to the CC_List sheet and get the next cost center (see Table 1).

```
Windows(MasterFile).Activate
Sheets("CC_List").Select
Activecell.Offset(1, 0).Select
```

First go to the cell containing the cost center that's just completed (the ActiveCell), and then move down one row to the next cost center.

The Wend statement comes next, telling the macro to go back to the beginning of the While loop and execute the statements again with the new cost center (the new ActiveCell). This will continue until all the cost centers have been processed and the blank cell at the bottom of the list is reached.

Once all the cost centers are processed it's time for the shutdown chores. First the output file is saved and closed.

```
Windows(SaveFile).Activate
ActiveWorkbook.SaveAs filename:=(DataDir & SaveFile)
ActiveWorkbook.Close
```

Then the data files are closed.

```
Windows(GLFile).Activate
ActiveWorkbook.Close
```

Finally, the End Sub statement tells the Do_It_All main macro to come to a glorious end.

Do the Sub-Macro

The Do_It_All main macro passes control temporarily to sub-macros that do specific jobs for each cost center. The Copy_GLData sub-macro copies external GL data into the main workbook. Begin the sub-macro with a Sub statement and then select the first cell in the external GL data file (Table 3).

```
Sub Copy_GLData( )
Windows(GLFile).Activate
Sheets("Sheet1").Select
Cells(6, 1).Select
```

Test to see if the first cell is equal to the CostCtr variable, and if not, move down one row and test the next cell. This continues until the CostCtr variable is found. Use the Do loop to repeat steps until a condition is met.

```
Do
    If ActiveCell = "" Then
        End
    ElseIf ActiveCell <> CostCtr Then
        ActiveCell.Offset(1, 0).Select
    End If
Loop Until ActiveCell = CostCtr
```

The above will find the CostCtr variable, or if it can't be found, then end execution so you can discover why the current cost center has no data.

Once you find the CostCtr variable in the GLData file, then copy its range of data.

```
Range(ActiveCell, ActiveCell.Offset(0, 5)).Select
Selection Copy
```

The range to copy is from the ActiveCell (the selected cell) to the cell offset by 0 rows and 5 columns. For example, A6:F6 for cost center 101 in Table 3.

The copy from range is pasted into the main workbook (Table 2).

```
Windows(MasterFile).Activate
    Sheets("GL_Data").Select
    Range("B7").Select
    Selection.PasteSpecial Paste:=xlValues, Transpose:=True
```

The sub-macro Copy_GLData has finished its duties. The End Sub statement returns control to the Do_It_All main macro.

Don't Forget to Save!

The Save_Results sub-macro saves the results for the current CostCtr before continuing onto the next cost center. You'll use the SaveFile workbook, which was created when the Do_It_All main macro first fired-up. Begin with a Sub statement, then go to the SaveFile workbook and create a sheet to receive the results.

```
Sub Save_Results ( )
    Windows(SaveFile).Activate
    ActiveWorkbook.Sheets.Add
    ActiveSheet.Name = CCSaveSheet
```

The variable CCSaveSheet comes from the cost center list on the CC_List sheet (Table 1).

The results for the current CostCtr are located in the main workbook on the Calculator sheet.

```
Windows(MasterFile).Activate
    Sheets("Calculator").Select
```

It's a good idea to tell Excel to re-calculate before copying any results.

```
Calculate
```

Mark the entire sheet for copying using the Cells object.

```
Cells.Select
Selection.Copy
```

Return to the SaveFile workbook and paste your results onto the CCSaveSheet.

```
Windows(SaveFile).Activate
Sheets(CCSaveSheet).Select
    Cells.Select
    Selection.PasteSpecial Paste:=xlValues
    Selection.PasteSpecial Paste:=xlFormats
```

The Save_Results sub-macro has finished its job. The End Sub statement returns control to the Do_It_All main macro so the next cost center can fire-up.

A final note on sub-macros. Sub-macros do the specific chores you need getting done. A big help in coding sub-macros is the Record function in the Visual Basic Editor. The Record function automatically writes the sub-macro code as you manually perform the keyboard and mouse operations. Copy the recorded sub-macro into the main workbook VBA module, substitute variable names where needed, and add a line in the Do_It_All main macro to activate it. Small modifications to the Do_It_All main macro will handle the logic needed to cycle through your list of items. In a nutshell that's how you can quickly modify the general framework described here for your own purposes.

The Moment of Truth

Finally, you're ready to make the whole thing work! Open the main workbook MasterFile, in my case MyMacro.xls. Select Tools on the toolbar, and then Macro. Highlight the Do_It_All macro and click Run.

And away she'll go!

Or maybe not. Murphy's Law will prevail over your carefully written code. It's unlikely your code will run the first time, or the second or third time, either.

When your code stops executing and the VBA error dialog box appears, click Debug to go to the line that's causing the problem. Fix it — always easier said than done — and try to run the Do_It_All macro again. (Excel 97 users: after fixing the problem reset the code in the VBA editor by clicking Run on the toolbar and then Reset.)

If you're persistent, sooner or later the Do_It_All macro will run. Your screen will light up as Excel flies through the commands you have given it with the VBA code. Although entertaining, the macro will run much faster (and I mean *much* faster) if you turn the screen updating off. Make this the first line in the Do_It_All main macro:

```
Application.ScreenUpdating=False
```

That's All, Folks!

Whew! That was a lot of stuff to go through. Was it worth it? Let's see what was accomplished with automation:

- Stepped through a list of items, in this case a list of branch cost centers,
- Retrieved data for each item from both financial and non-financial systems (for brevity only GL data was retrieved here),
- Performed complex spreadsheet calculations on the imported data, and
- Prepared a report and saved the results for each cost center.

Macros allow you to economically combine data from disparate sources. Macros allow you to do the analysis you've avoided in the past due to excessive amounts of manual effort. Macros allow you to conduct repetitive analyses, such as trend analysis and what-if sensitivity analysis. Investing ten to twenty hours climbing the VBA learning curve can return automation that performs valuable chores for you. Chores that make once impossible analysis possible. Chores that help you understand what makes your company tick. Chores that make you the hero.

So, you decide. What is that worth?

RULES OF THE VBA ROAD

(1) Use an apostrophe ' to make comments on a line or part of a line. Use comments generously throughout your code to explain what's going on.

```
'This is a comment
```

(2) Indent, Indent, Indent! Use indentation to outline the logic in your code. VBA ignores white spaces, such as indentations, so indent liberally to make your code easier to understand and debug.

(3) Always declare variable and constant names before use. Just add new variable names to the variable list at the beginning of the macro.

```
Dim CostCtr as String
Private Const DataFile = "1Q99Data.xls"
```

(4) Use quotes around a fixed object, such as the tab on a sheet.

```
Sheets("1Q99Data").Select
```

(5) Do not use quotes around a variable name.

```
Dim DataSheet as String
DataSheet = "1Q99Data"
Sheets(DataSheet).Select
```

(6) Always use Option Explicit, which forces you to declare variable or constant names before use. This eliminates the possibility that you accidentally misspell a variable or constant name.

```
Option Explicit
Private Const DataFile = "1Q99Data.xls"
Dim DataSheet as String
```

(7) Use a space and an underscore _ to continue your code on the next line. It is easier to read and debug your code when it is viewed on one screen width.

```
Activecell.Value = Application.Sum(Range _
    (Cells(BegRow, ColNum), Cells(EndRow, _
    ColNum)))
```

AMIfS
Association for Management Information in Financial Services

Journal of Bank Cost & Management Accounting

Re-Think Customer Segmentation for CRM Results
—ROBERT GILTNER and RICHARD CIOLLI—

Better Financial Planning with Balance Sheet Modeling
—STEPHEN EASTBURN—

Zero-Based Staffing™ for Performance Improvement
—ADAM ISLER—

Excel Macro Combines Financial and Non-Financial Data
—DAN HILL—

The Forsaken Side of Risk Management: Have Deterministic Approaches Gone Too Far?
—W. RANDALL PAYANT—

Table of Contents

ASSOCIATION FOR MANAGEMENT INFORMATION IN FINANCIAL SERVICES