



10 Must Have Excel Power User Tips for Accountants and Financial Analysts

Provided by Resource Planning Solutions

21031 Ventura Blvd

Suite 415

Woodland Hills, CA 91364

(818) 436-0781

www.rpscgi.com

Table of Contents

Introduction	3
Tip #1 -- How to efficiently manipulate text strings with Excel	4
Tip #1 -- Flexible Worksheet Consolidation	8
Tip #2 -- Using Defined Names	12
Tip #3 -- Loading data from external sources	19
Tip #4 -- Leveraging Excel's VLOOKUP() function	26
Tip #5 -- Using linked lists in dialogs and data validation to control user input	32
Tip #6 -- Setting up simple user controls	42
Tip #7 -- Dynamic reporting tools	49
Tip #8 -- Using customized lists to increase productivity	56
Tip #10 -- Creating User Defined function macros	59
Summary	67

Introduction

At Resource Planning Solutions, we are focused on delivering financial performance management solutions to our clients' that fulfill their requirements for financial planning, financial reporting and business analytics. Successful implementation of our solutions is based on the following sequence of activities: 1) work with clients to help them better understand the availability and value of their financial and operational data; 2) develop a prioritized plan to harness the latent potential within each client's financial and operational data; 3) increase organizational alignment to financial and operational plans through implementation of business metrics; and 4) establish or improve forecasting processes that utilize insight gained through the analysis of business metrics to proactively influence financial outcomes.

We strive to transform our clients' view of their financial and operational data into that of a strategic asset. Through this transformation, clients begin to understand the potential opportunity to influence and improve operational performance through utilization of information that is frequently already available to them. We then work collaboratively with operational decision makers to develop a prioritized plan to access, integrate and consolidate financial and operational data to achieve near-term goals. We utilize a host of tools, including Excel, Access, third-party applications, to cost effectively implement our solutions.

With the implementation of new processes, we then work with clients to segregate their key performance indicators. These indicators are then incorporated into dash boards that are used to ensure that execution is tightly linked to operational planning. As a result of reviewing operational performance activity, clients are able to gain improved insight into the cause-and-effect relationships affecting business performance; that insight is then used to develop improved forecasting based on their business drivers. We then help our clients incorporate those business drivers back into the planning and forecasting processes to ensure that newly gained intelligence continues to influence financial outcomes. Our structured process guarantees that our financial performance management solutions deliver results.

We produced our *10 Must Have Tips* list as a way to help both accountants and financial analysts acquire some new perspective on problem solving with Excel. We have spent hours and hours working with clients, helping them deploy Excel-based solutions to a wide range of financial planning, reporting and analysis tasks. Many of the tips illustrate skills that can be put to practical use on a daily basis, such as the tips on parsing text data and how to effectively use VLOOKUP(). Other tips, such as the one on setting up simple user controls, may be used less frequently, but can substantially increase the usability of a workbook application; consequently we added it and a couple of other similar tips to the list.

Though our firm uses Visual Basic extensively, our initial thought was to avoid its use in our *10 Must Have Tips* list. But, this document does include a couple of tips relating to Visual Basic, because it is such a critical skill for any Excel power user. This is because spreadsheet applications that utilize Visual Basic contain vastly greater capability than those that do not. Readers should review those tips and experiment with their own ideas. Hopefully, readers will want to take the next step and learn more about the subject.

We hope you will find the tips contained in this document helpful. These techniques get used over and over again in our consulting practice; consequently we understand the value that these ideas can deliver. If you have any feedback, thoughts on improvements or additional capabilities you would like to see added, please send us an email at Info@rpscgi.com.

Tip #1 -- How to efficiently manipulate text strings with Excel

Background

An issue that we frequently come across with our clients is users that are a little overwhelmed when they need to rearrange text strings differently than from the way that the text string was imported from a source system. A classic example is when someone has downloaded a list of names from an HR database and the name was exported as "Last Name, First Name" and the user wants it "First Name Last Name" in a single cell (or sometimes it needs to be parsed into two columns—first name, last name). This happens with all kinds of data that is exported from accounting systems, CRM systems, HR systems, etc. As long as the text string is delimited, such as with a comma, asterisk, blank space, or colon we can use Excel's different text functions to get the text string parsed as required.

Application

There are five primary text functions that users need to learn, which are LEFT(), RIGHT(), FIND(), LEN() and MID(). Here is how each of them work:

LEFT() – This has only two possible parameters: LEFT(*source, # characters*). The source is the text cell to be parsed, and the # characters are the number of characters we want returned beginning from the left most character. For example, at cell D10 we have the word "MOUSE". The following statement =LEFT(D10,4) will return MOUS (the left-most four characters).

RIGHT() – This works exactly the same as the LEFT(), just beginning from the opposite side of the text string. So RIGHT(D10,2), using the same example as above would return SE (the right-most two characters).

FIND() – Like the previous two functions, this function only requires two parameters: FIND(*character(s) to be found, source string*). The first parameter needs to be enclosed with double quotes and represents the string to be found. The second parameter is the cell address of the text string to be searched. The FIND() function will return the position number, within our string, of the first occurrence of the character(s) being sought. Though this function only requires two parameters, there is a third parameter that is optional, which we will describe once we have gone through the basics. Using our "MOUSE" example again, FIND("O",D10) would return the number 2. The letter "O" in MOUSE, is the second character in our text string.

LEN() – The LEN() function returns the length of the referenced text string. If we have "The quick brown fox jumped over the lazy dog" in cell B2, then =LEN(B2) will return 44 which is the number of characters in our text string.

MID() – The MID function has three required parameters: MID(*source, starting position, number of characters*). The MID() function will extract a number of characters from within a text string. Again using our previous example assuming D10 = "MOUSE", then MID(D10,2,1) would return O, which is the second character in our text string. This is because we asked the function to start at the second character, which is "O", with the "2" at the second parameter, and then asked for only one character with the "1" at the third

parameter. If we had MID(D10,2,2), the function would return "OU", MID(D10,2,3) would return "OUS" and so on and so forth.

Using our initial "last name, first name" example, we can illustrate how these different text functions work together. So assume that we have received a list from our IT department with the employee names formatted "Last Name, First name" and that we need to parse the names into separate columns (this would be required if we were going to use the data in a mail merge application that required a first name salutation). We have already imported our "Last Name, First Name" data into column D, beginning at row 10. Now we need the last name in column E and first name column F. We will use the comma as a delimiter. Typically when names are provided this way, such as in "Dean, Jimmy" there is also a blank space to contend with after the comma.

Step 1: Parse out the last name — Using the MID and FIND functions together, we just need to know where the comma is in our string and then subtract one from the number returned from FIND(), and that will be the number of characters needed to parse the last name. The last name starts at the first character so we only have to provide the number "1" as the correct start location. With our text string example located at D10, we want the last name parsed to E10, which would be:

E10 = MID(D10,1,FIND(",",D10)-1)

Breaking that down, assuming our first text string is Dean, Jimmy at D10, we tell the MID function to start extracting at the first character, which is the "D" in Dean. The FIND function is going to look for the first occurrence of a comma in the text string and return the numeric position in the text string, which would be then number 5 ("D" is one, "e" is two, "a" is three, "n" is four and "," is five). You will note that we subtracted 1 from the FIND result with FIND(",",D10)-1 which is going to return one less than the string position where the comma is located or the number four. Excel resolves our MID function to:

E10 = MID(D10,1,4) = Dean

Voila, we now have the last name parsed into cell E10.

To parse the first name into cell F10, we need to introduce one more function, which is LEN(). The LEN() requires only one parameter, which is a text string reference. In our example string, "Dean, Jimmy" is eleven characters long (do not forget the blank space after the comma); therefore, LEN(D10) would return 11. Now we have everything we need to parse out the first name.

At F10, we will add the following function =MID(D10,FIND(",",D10)+2,LEN(D10)). We know that looks a little intimidating, but hang in there for another minute. Again, the FIND(",",D10) is going to return 5—the position of the comma—and then we need to add 2 to that position to get to our starting point for the string extraction, which is character string number seven ("D" is one, "e" is two, "a" is three, "n" is four, "," is five, " " or blank space is six, and "J" is seven, which is where we want to start). The LEN(10) is going to return the total length of our string, which is okay. We do not have to have the precise number of characters to the end of our string in this example. We just need to make sure that we are going to extract all the characters to the end of the string. Excel resolves our second function to:

F10 = MID(D10,7,11) = Jimmy

Now that we have the last name and first name parsed into separate columns, we just need to copy and paste our formulas down through all the rows and our entire list is parsed.

A little more complex example

Now that we are through the basics, we can discuss how to nest these functions together to parse more complex strings. In this example, assume that we receive a work order completion report from our ERP system that includes the assembled part number, a lot number, and the quantity produced. Our ERP system uses "*" to delimit the three pieces of information, which would look like 7053312-007*154789513*376 for material number 7053312-007, lot number 154789513, with a completion quantity of 376. Since we are in Finance, we are only interested in the part number and quantity completed (the Quality department can worry about the lot number). We import the report into Excel, with the first record starting at D10 again. The final result needed is to parse the text string such that the material number ends up at E10 and the quantity at F10. In our example, the material number and lot numbers can vary in length, so we will need some logic to correctly parse out the completion quantity; otherwise we could just parse the text string with two MID() formulas.

The approach to parse material number is similar to when we parsed "Dean, Jimmy" since we can start at the first character. We just need to use "*" as our delimiter rather than ",". Using the MID() and FIND() functions together to parse out the material number, we will see the following:

E10 = MID(D10,1,FIND(" ",D10)-1) = 7053312-007

Getting to the quantity is a little more complicated. We mentioned earlier that the FIND() function has an optional third parameter. The optional third parameter tells the FIND() function where to start searching for our "character(s) to be found". The FIND() function parameters are as follows:

FIND(*character(s) to be found*, *source*, *starting character*)

If *starting character* is omitted, then the FIND() function begins looking at the first character in our text string. Since we are looking for the second "*", which is where the completion quantity starts, our FIND() needs to use the third parameter to find the correct location within our string to initiate the search, which is the first character past the first "*". To accomplish this, we are going to nest our FIND() functions. The following FIND() will locate the position of the second "*":

=FIND(" ",D10,FIND(" ",D10)+1)

Remember our example string is 7053312-007*154789513*376. The third parameter of the first FIND() function FIND(" ",D10)+1, is initially going to locate the first "*" at string index 12, and then add 1 for a string index of 13. At text string index 13, we have the first character or "1" in the lot number field. But this is the first character past the first "*", which is what we wanted. So then our first FIND() function resolves to the following:

=FIND(" ",D10,13)

The second FIND() will begin searching for the second "*" at the 13th character index and will return 22, the location of the second "*". We then add one to our FIND() result and use

MID() to capture the completion quantity. The whole formula required at F10 to capture the completion quantity is as follows:

F10 = MID(D10,FIND("*",D10,FIND("*",D10)+1)+1,LEN(D10))

The above formula will then resolve to the following:

F10 = MID(D10,23,25) = 376

Here, we only used one level of nesting. If needed, we could create more nesting if there were more delimiting characters to navigate around. But at some point there is going to be a tradeoff between reliably parsing text strings with Excel functions and just writing a function macro to handle more complex tasks. Tip #10 deals with just such a situation.

Tip #2 – Flexible Worksheet Consolidation

Background

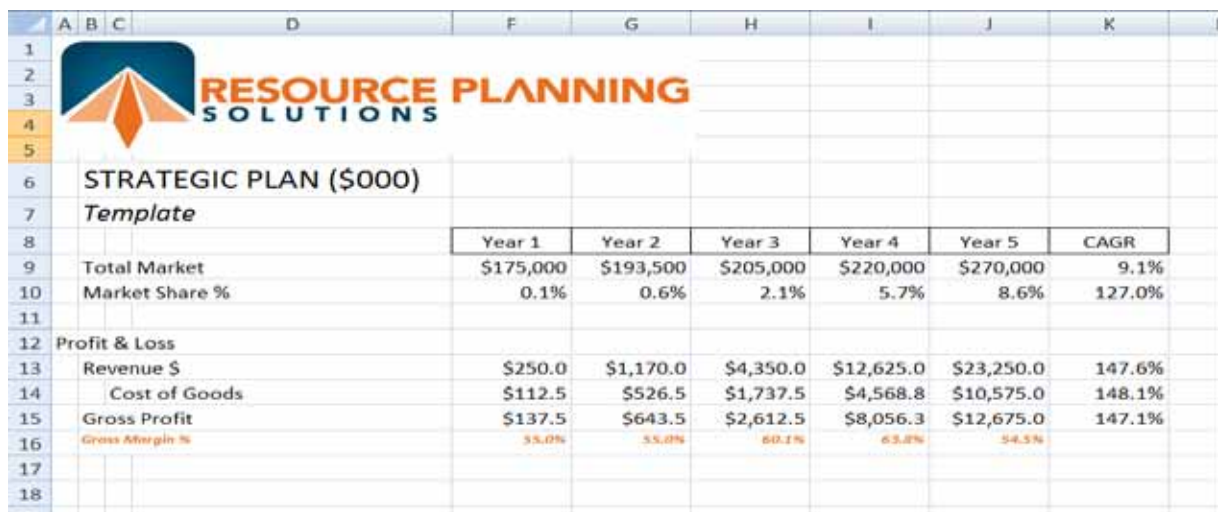
Though Excel provides built-in functionality to consolidate worksheets, it is a bit overly complex for most situations. The solution that we frequently utilize does consolidate worksheets, without using Excel's consolidation function, while also giving users more flexibility to change what is being consolidated.

Typically, we will create a series of financial statements that need to be consolidated into a consolidated enterprise. [Note: Because we typically are consolidating financial statements, most of this tip will be referring to the consolidation of financial statements.] The supporting financial statements may represent lines of business or divisions or some other entity type. By using our approach to consolidate worksheets, we can instantly modify the composition of our consolidation by either adding or removing worksheets from the consolidated total.

The one limitation to this approach is that each worksheet needs to be identically structured relative to the items that are going to be consolidated. Generally this requirement is not particularly onerous given the nature of worksheet consolidation. In addition to the line of business-level worksheets, we will need one worksheet that sums all the lines of business as a consolidated financial statement.

Application

To illustrate, assume that we have a business consisting of three lines of business (LOB) titled LOB1, LOB2 and LOB3. Each LOB's P&L and balance sheet is additive to the consolidated financial statement. In our example, the leadership team is considering adding a fourth LOB and wants to be able to easily consolidate or exclude the fourth LOB, to better analyze LOB4's impact on the consolidated financial statements. This will require five worksheets, one for each of the four LOBs and one for the consolidated business. We start by creating a template for our financial statement with the years going across the columns. The template looks as follows:



	Year 1	Year 2	Year 3	Year 4	Year 5	CAGR
STRATEGIC PLAN (\$000)						
<i>Template</i>						
Total Market	\$175,000	\$193,500	\$205,000	\$220,000	\$270,000	9.1%
Market Share %	0.1%	0.6%	2.1%	5.7%	8.6%	127.0%
Profit & Loss						
Revenue \$	\$250.0	\$1,170.0	\$4,350.0	\$12,625.0	\$23,250.0	147.6%
Cost of Goods	\$112.5	\$526.5	\$1,737.5	\$4,568.8	\$10,575.0	148.1%
Gross Profit	\$137.5	\$643.5	\$2,612.5	\$8,056.3	\$12,675.0	147.1%
Gross Margin %	55.0%	55.0%	60.1%	63.8%	54.5%	

The above worksheet is the template for our consolidated financials as well as for the four LOBs with year 1 starting in column F, and the total market for each LOB starting at row 9. Each of the LOBs and consolidation worksheets need to be structured the same, for all cells to be consolidated. So if the 2011 revenue forecast for LOB1 starts at LOB1!F13, with the 2012 forecast at LOB1!G13, then it will be the same for all the other LOB worksheets, as in the following:

<u>Worksheet</u>	<u>2011</u>	<u>2012</u>	<u>2013</u>
Revenue LOB1	LOB1!F13	LOB1!G13	LOB1!H13
Revenue LOB2	LOB2!F13	LOB2!G13	LOB2!H13
Revenue LOB3	LOB3!F13	LOB3!G13	LOB3!H13
Revenue LOB4	LOB4!F13	LOB4!G13	LOB4!H13

Given the above format, you could easily consolidate the 2011 forecast with the following formula on the consolidation worksheet:

2011 Consolidated Revenue = LOB1!F13 + LOB2!F13 + LOB3!F13 + LOB4!F13

The issue that arises is when you want to remove one of the LOBs from the consolidation, which means that every formula referencing LOB4 needs to be edited. Our approach to flexible consolidation eliminates the need to individually reference each worksheet cell as was done above. In order to make use of the flexible consolidation approach you will need to add two extra worksheets that establish the range. We usually name the two worksheets "Start" and "End". The names can be whatever you like, but should be descriptive. Once the "Start" and "End" worksheets have been created, we insert the individual LOB worksheets between them. When you have created the five worksheets (Summary, LOB1, LOB2, LOB3 and LOB4), and two bookends (Start & End), the ordering of the worksheets needs to appear as follows:



The "Summary" worksheet is where all the worksheets will be consolidated. In order to SUM the four LOBs, you simply need a formula that can SUM across all the worksheets from "Start" through to "End". With the 2011 revenue projections for each LOB at F13 on each of the four worksheets, the 2011 revenue can be consolidated on the Summary worksheet at Summary!F13 with the following formula:

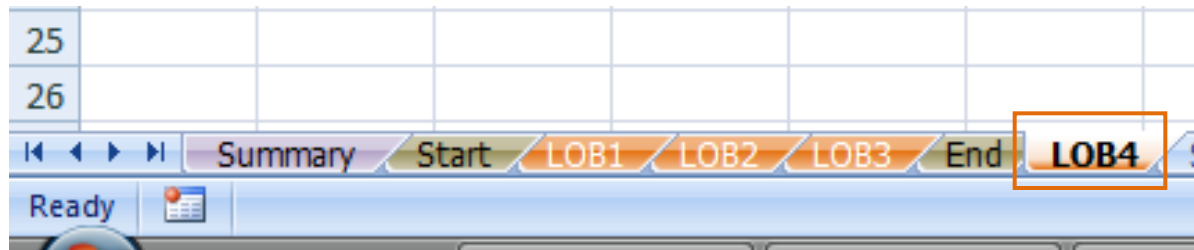
What is being consolidated: Year 1 revenue for all LOBs

At Cell Address: Summary!F13

Formula: =SUM(Start:End!F13)

Our new formula--SUM(Start:End!F13)--is equivalent to =Start!F13 + LOB1!F13 + LOB2!F13 + LOB3!F13 + LOB4!F13 + End!F13. But the new formula is all we need to

summarize all four LOB worksheets onto the Summary worksheet. Now, if we are asked to display the consolidation without LOB4, we simply drag the LOB4 worksheet beyond the “Start” and “End” bookend range. Once we move the LOB4 worksheet beyond the “End” worksheet, the worksheet tabs now appear as follows:



With LOB4 outside the SUM formula range, the values at Summary!F13 will only include LOB1, LOB2 and LOB3. And voila, LOB4 is now excluded from the consolidation, without having to adjust a single formula. This approach will allow us to create some very sophisticated reporting and analysis spread sheets, while permitting addition and deletion of worksheets within our consolidation, with much less formula editing.

Our example is rather simplistic. This approach can be used in larger consolidation. Assume that we have to restructure a sales force with 400 Territories. Each Territory tiers into a Region, of which there are 20. Each Region tiers into a District, of which there are four. The four Districts tier into the total company. Though this is rather large, if each territory is a worksheet, our approach would permit the required flexibility needed to restructure the entire sales force. The worksheets might be organized as follows:

Region1Start
 Territory001
 Territory005
 ...
 Territory150
 Region1End

Region2Start
 Territory250
 Territory305
 ...
 Territor395
 Region2Emd

Through all 20 regions

District1Start
 Region1Summary (This summarizes Region1Start:Region1End)
 Region2Summary (This summarizes Region2Start:Region2End)
 ...
 Region7Summary (This summarizes Region7Start:Region7End)
 District1End

District2Start
 Region9Summary (This summarizes Region9Start:Region9End)
 Region10Summary (This summarizes Region10Start:Region2End)

...
Region15Summary (This summarizes Region15Start:Region15End)
District2End

Through all four districts

TotalCompanyStart
District1Summary (This summarizes District1Start: District1End)
District2Summary (This summarizes District2Start: District2End)
District3Summary (This summarizes District3Start: District3End)
District4Summary (This summarizes District4Start: District4End)
TotalCompanyEnd

TotalCompany Summary (This summarize TotalCompanyStart:TotalCompanyEnd)

With this structure, without having to manipulate formulas, we could change the composition of our regions and districts very efficiently with only our mouse.

The one caveat to this approach is that the bookend worksheets (Start and End) are part of the SUM function, which means those worksheets must be blank; otherwise the SUM function will include any data on those two worksheets.

Tip #3 -- Using Defined Names

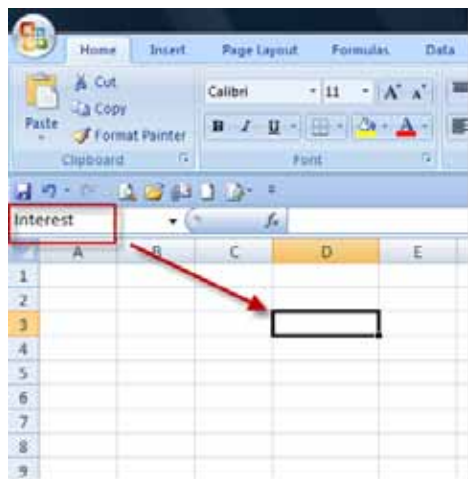
Background

One of the things that make spreadsheets so powerful, the ability to link workbooks and worksheets, is also a primary source of errors for many of our clients. For example, we see users linking their revenue worksheet to their accounts receivable worksheet, since revenue is typically an input to accounts receivable forecasting. At the outset, everything is great. But inevitably another user will come along and add a row to the revenue forecast. Originally, total revenue was at row 20 and now it is at row 21. Unless the accounts receivable file was open when the row was added to the revenue forecast workbook, the next time a user goes to update the accounts receivable file, their accounts receivable forecast will link to the incorrect row (row 20 on the revenue worksheet). To reduce our vulnerability to broken links, we have clients use Defined Name ranges. An added benefit to Defined Name ranges is that formula maintenance becomes easier.

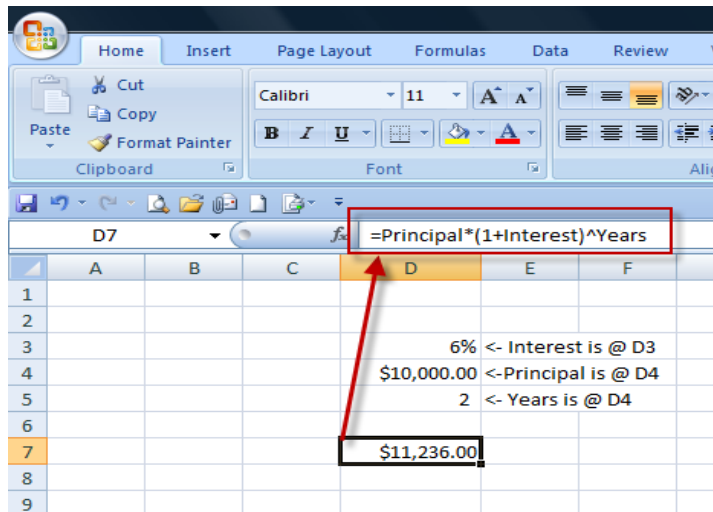
Application

Excel provides users the ability to create Defined Name ranges on a worksheet. A Defined Name can refer to a single worksheet cell, contiguous range of cells along a row or column (one-dimensional array) or contiguous group of cells along multiple rows and columns (two-dimensional matrix). Let's start with a Defined Name cell.

We can very easily create a Defined Name for a single cell and then use that name as a reference in formulas as a constant. Assume that we need to generate compound interest calculations and that we want to use Defined Named references for our three variables, "Interest," "Principal," and "Years". The "Interest" rate will reside at cell D3, "Principal" amount at cell D4 and number of "Years" at cell D5. There are a couple of ways to create Defined Names for our cells. The first method, with the active cell being the one that we are about to name (cell D3 in the example below), we type the Defined Name—Interest--into the Name Box highlighted below:

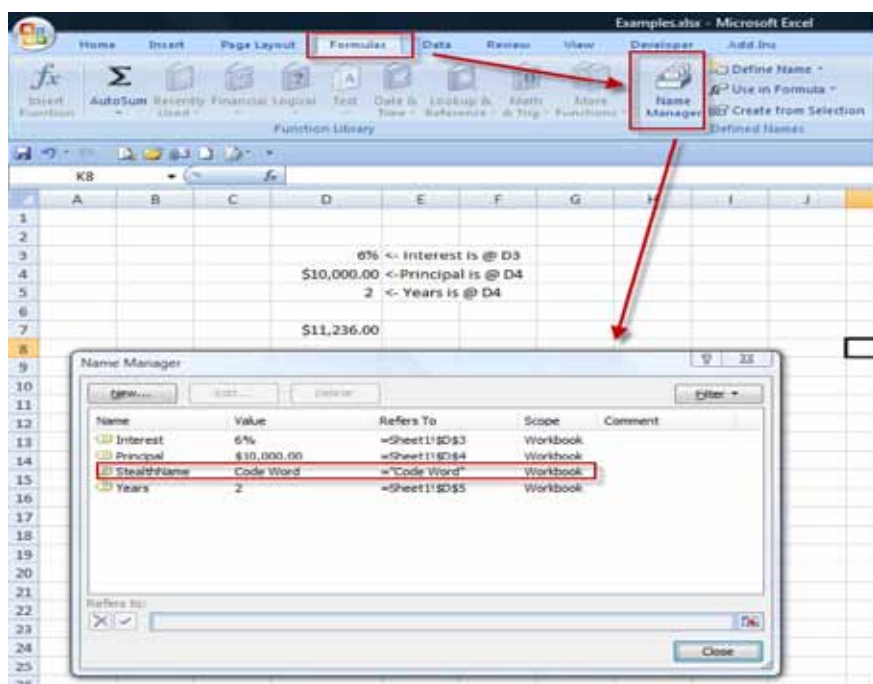


We can then do the same for "Principal" and "Years," in cells D4 and D5 respectively. Then at cell D7 we can input our formula. Our formula for calculating compound interest is $(P \times (1 + R)^N)$. Without using our Defined Name references, the formula for calculating this amount would be $=D4*(1+D3)^{D5}$. But, with our Defined Name references, we create the following at cell D7:



There are many different uses of Defined Name references that can make Excel more manageable.

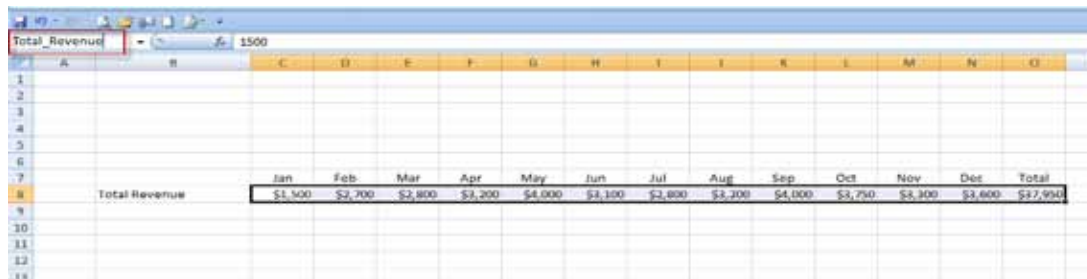
To see the Defined Name references we created, we need to navigate to the Names Group on the Formulas tab. Select the Name Manager to get the dialog box referenced below.



In the Name Manager dialog box you can see the three named cells we created earlier, their physical locations on the worksheet and assigned values.

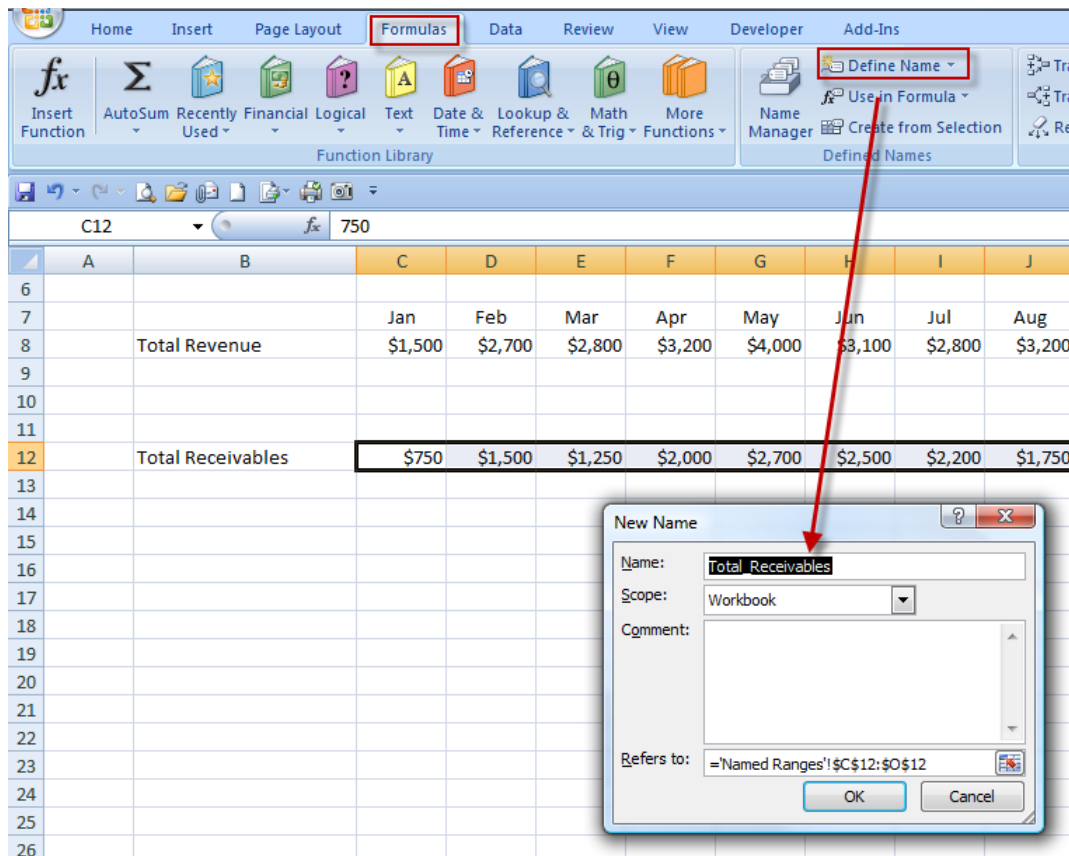
In addition to naming cells on a worksheet, we can create Defined Names that do not reference a cell on your worksheet, but exist as part of the worksheet. The “StealthName” Defined Name reference above is an example of a constant that does not exist on any Worksheet. The named constant is set to the text string “Code Word”. We can set it to a value, text string or logical value. Typically, we use a this type of constant to create values we want hidden from users of our application once a workbook has been protected. We can only change the value of this type of constant through the Name Manager or through a macro. It cannot be modified directly on a worksheet.

Defined Name ranges are very useful in any number of applications. There are two ways to create a Defined Name range. The first is using the same process described in the single cell example above. First, highlight the desired range of cells, either along a row or column, depending upon what works best for you. The next step is to type in the Defined Name in the Name Box indicated below (see Total_Revenue below). In this example, we are going to highlight the row where total revenue is located at row 8, columns C through N, with a total at column O.

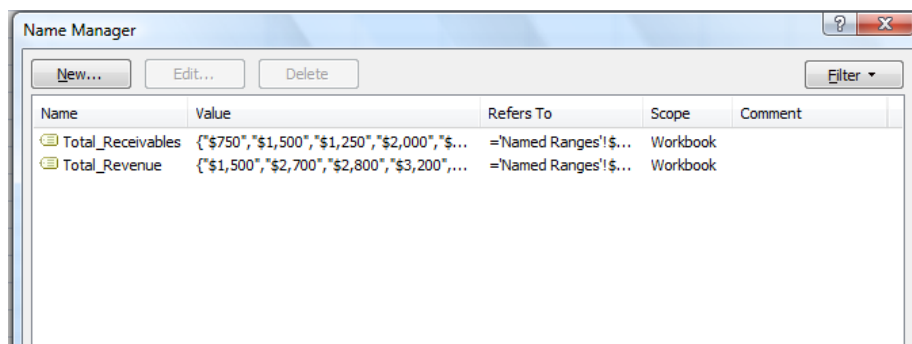


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2															
3															
4															
5															
6															
7			Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
8		Total Revenue	\$1,500	\$2,700	\$2,800	\$3,200	\$4,000	\$3,100	\$2,800	\$3,300	\$4,000	\$3,750	\$3,300	\$3,600	\$37,950
9															
10															
11															
12															
13															
14															

The second way to create a Defined Name range is using the New Name function referenced below. Before selecting Name Range, you should highlight the cells that will become the one dimensional matrix. In this second example, Total Receivables are referenced at row 12, columns C through N. As you can see below, Excel used the row title (“Total Receivables”) to suggest a name for our new range. We just selected “OK” to accept the Excel’s suggested name.



So now we have two named ranges, Total_Revenue and Total_Receivables. You can verify that the new Defined Name ranges exist by checking the Name Manager, which appears as follows:



From the Name Manager, we can see the name ranges as well as what values have been assigned to the ranges from our worksheet.

The advantage to using Defined Name ranges is that we can reference a worksheet range, in a completely different workbook, by using a named reference rather than a cells' absolute address. By referencing the named ranges rather than absolute addresses, the physical

location of the named ranges can move, e.g. a user adds a row at row 3, while the data reference by our named range remains intact. Now if a user adds a row at row 3, Total_Revenue can still be referenced correctly by another workbook even though Total_Revenue will now physically exist at row 9 rather than where we originally created Total_Revenue, which as at row 8. This vastly reduces the opportunity for broken links in our applications.

To reference or use a Defined Name range in a formula, we will need to use the INDEX() function. The INDEX() function has two arguments, the first being the Defined Name range, the second being the index number within the range. Total_Revenue has 13 elements, starting with the January value at index 1, ending with total revenue value at index 13. For example, the March revenue figure can be accessed using the following:

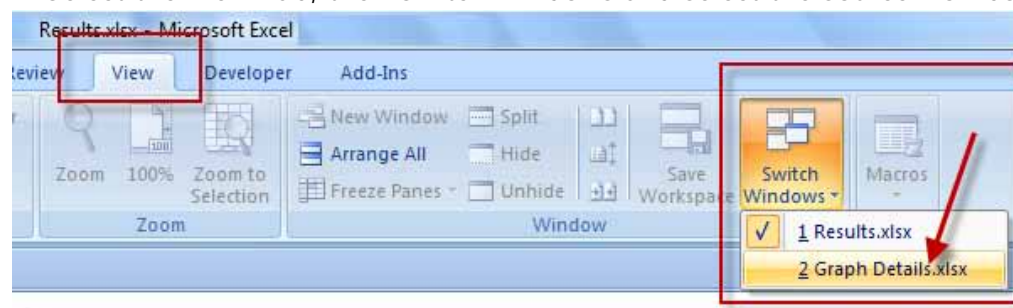
=INDEX(Total_Revenue,3) which will return \$2,800

In application, assume that we have two workbooks, one titled Graph Details.xlsx and the other Results.xlsx. The data for our graph is contained in Graph Details.xlsx and the graph itself will be in Results.xlsx. Since we frequently modify the Graph Details.xlsx workbook, we want to use a Defined Named range to reference the data we supply to the graphs in Results.xlsx. We will need to have both the source workbook (Graph Details.xlsx) and the destination workbook (Results.xlsx) open when the linkage is created. Navigating to the destination workbook, where our revenue graph will be created, we will want to add the first INDEX() function for the January revenue at cell C24 as follows:

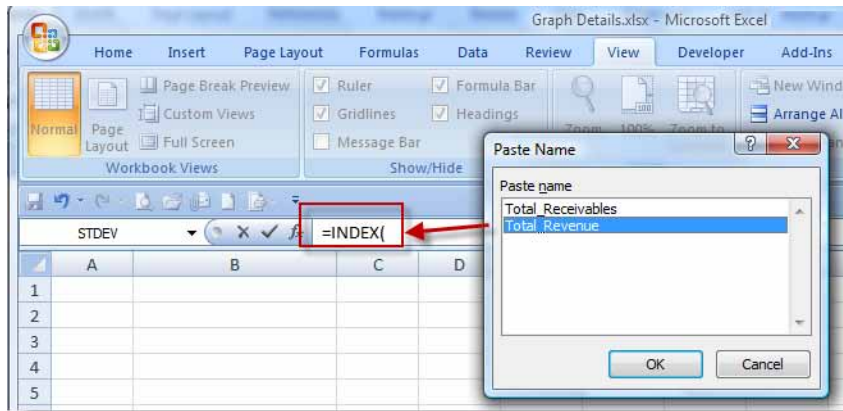
[illegible]

To paste in the external Defined Name reference we will need to follow these steps:

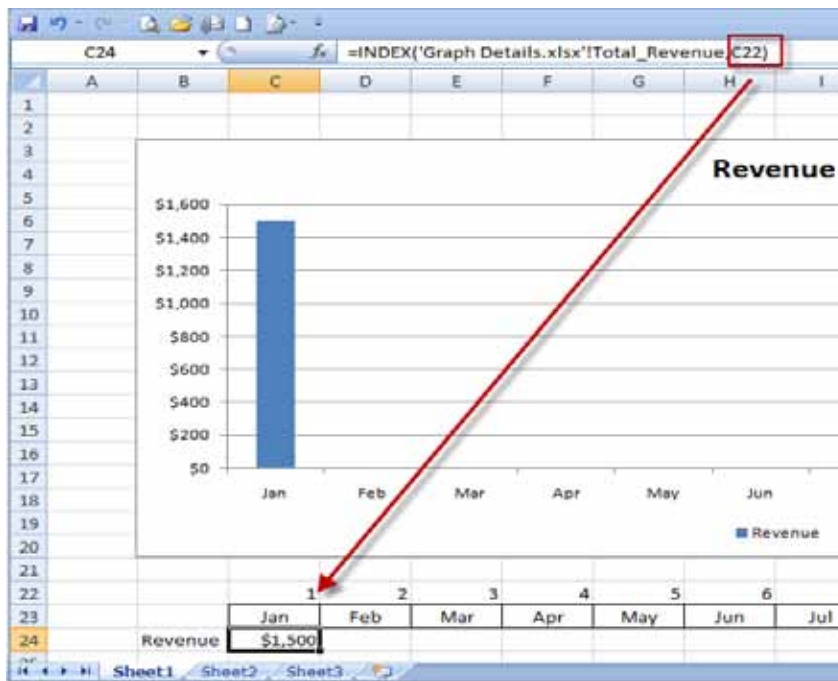
- 1 Select the View Tab, then Switch Windows and select the source workbook



- 2 Once we are on the source workbook, we hit the F3 key to initiate Excel's Paste Names function. We select the required Defined Name—Total_Revenue—and hit OK.



- 3 We add the index number that needs to be returned as the second parameter to the INDEX() function. In our example the index number required to return the January Revenue is the value 1. In this example we have put all the index references above the months so that we only need to reference the cell and can copy the INDEX() function across all the months. We can hide the index row later, if needed.



With the first formula created at C24 and the index references at row 22, we can paste the formula across the range to complete the graph. Once we have pasted the INDEX() formula we have the following result:



To prove the value of this approach, we need to close both workbooks. Then we reopen the Graph Details file and add several rows above the Total Revenue row, and Save/Close the file. Next we open the Results.xlsx file and the graph should remain the same since the INDEX() function references the Defined Name ranges rather than the cell addresses. This process can use this to great advantage when creating workbooks that will be modified by other users. Now we can allow users to add rows, still be able to reference the totals via an external link. Take a look at the other reference functions, such as MATCH(), to see how these other functions can extend the use of Defined Name ranges. Two-dimensional ranges are also available in Excel, but are beyond the scope of this document. Two-dimensional Defined Name ranges are better used as part of a Visual Basic application.

Tip #4 -- Loading data from external sources

Background

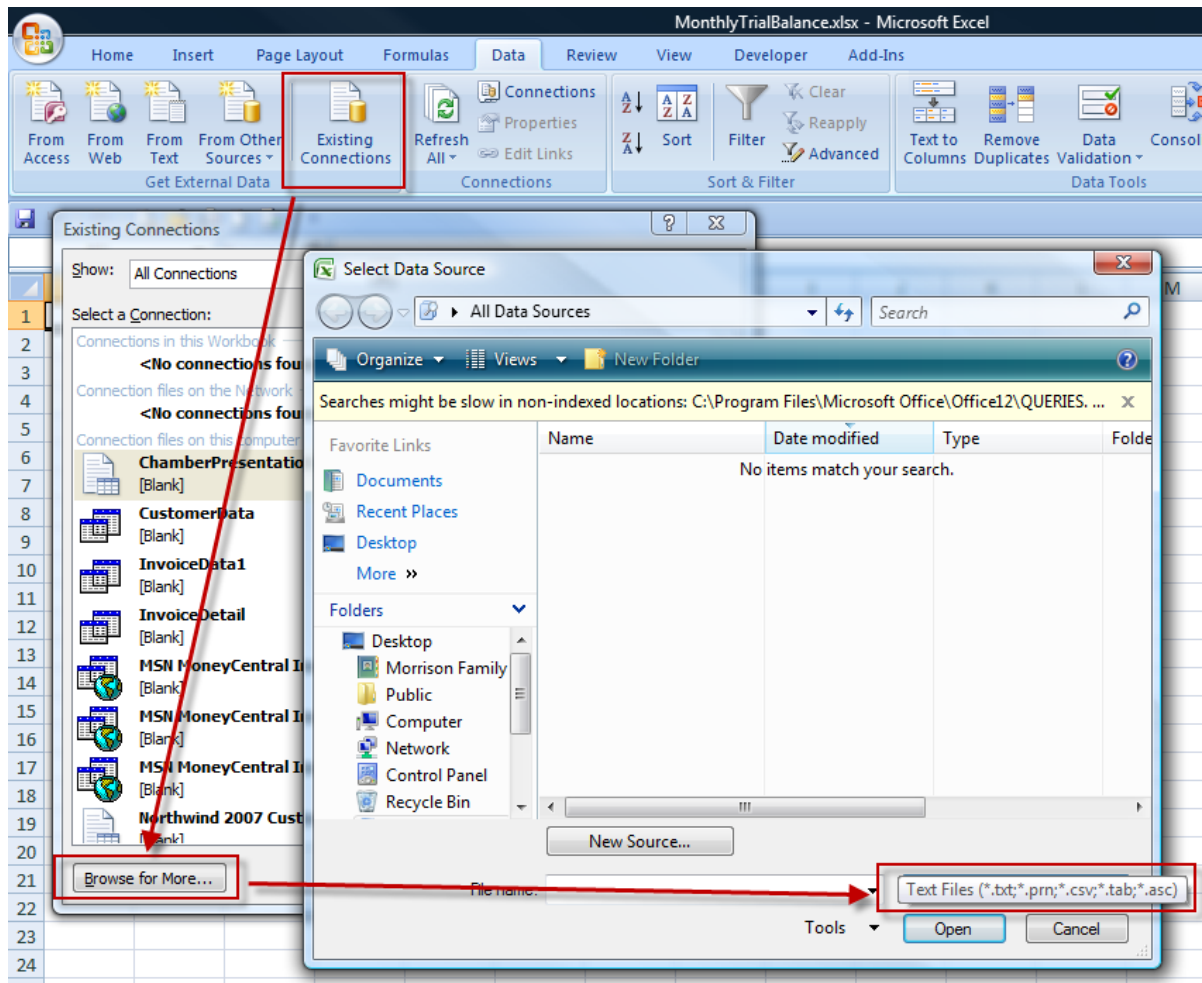
In our consulting practice, we frequently see users importing fixed width (or un-delimited) files using Excel's File/Open dialog. For occasional file imports, this practice is satisfactory. But, users that import the same fixed width file using Excel's File/Open dialog, over and over again, week after week, can be impacted because this is a very inefficient practice. The inefficiency is a function of having to define where columns are to be split, and then having to change the default data type Excel assigns to your fields, when the General type will not suffice (usually there will be a field that Excel will convert to numeric that should remain as a text field) with each import.

There is a much more efficient way using Excel's Workbook Connections functionality. You can easily set up a connection to a fixed width text file. Once it is set up, users can update their workbook by just refreshing the connection, which alleviates the need to tell Excel where to split the columns and what data type to apply to the various columns. We just open the connection and perform a Refresh with the latest file, and viola the result is correctly parsed every time.

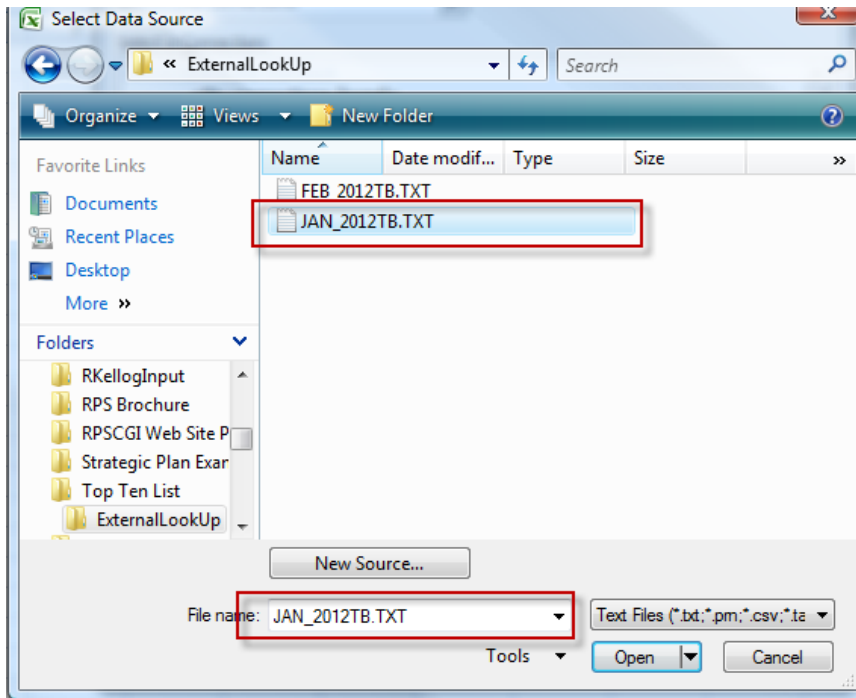
Application

In this example, we hypothetically download a trial balance from our accounting system monthly to analyze the general ledger accounts. The file is distributed as a fixed width text file, with account, account description, current-month debit and credits, year-to-date debits and credits as well as an inception-to-date net account balance.

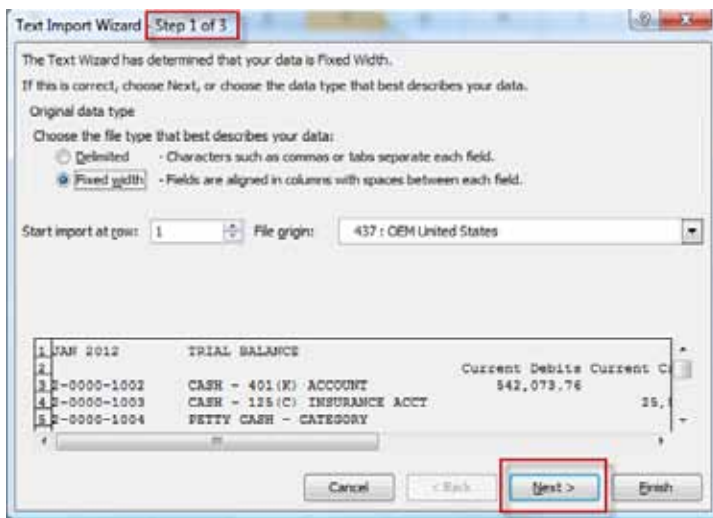
The first step is to create our connection to our fixed width text file. We open a new workbook and save it with the report name, which in this case will be MonthlyTrialBalance. Then we go to the Data menu tab and select Existing Connections (which is counterintuitive since we are creating a new connection). Selecting Existing Connections will bring up the Existing Connections dialog. At the lower left corner of the Existing Connections dialog, we will want to select Browse for More... This will bring up the Select Data Source dialog, which is where we can create a new connection. Change the file type to include *.txt files and then we navigate to our file.



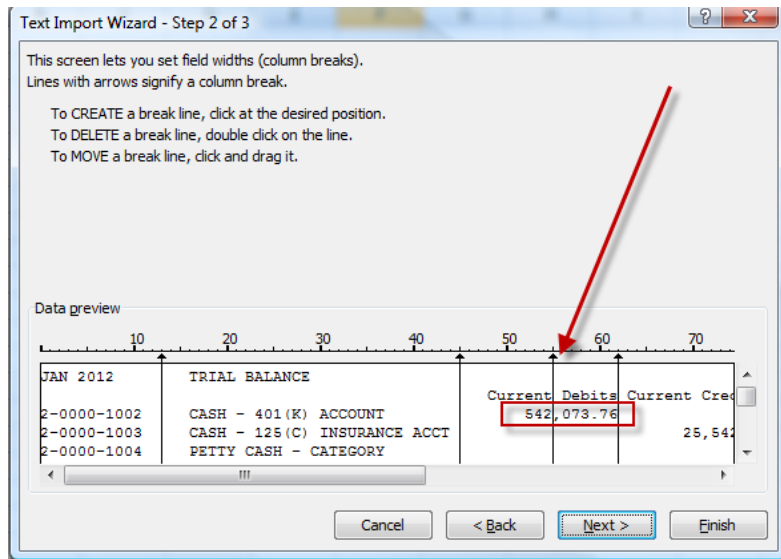
Once we have navigated to our file, we go ahead and select Open to establish the file import format. We will only need to perform this step with the initial import. The Connection Manager will remember all the formatting options we are going to specify. Once the file type is set to Text Files, we can now see our downloaded Trial Balance files in the Select Data Source dialog below. The January file, currently highlighted, will be opened first.



Upon initially opening our new text file, the Text Import Wizard will start. Typically there is not much to revise at the first step. The one caveat is to make sure that the correct radio button is selected, depending on whether the file is Delimited or Fixed Width. In our example it is Fixed Width. We go ahead and select Next.



This will bring up the second step, which allows us to specify how Excel will parse the fixed width file into columns. Once we specify the width of our columns, using the separators, we scroll up and down the file checking to ensure that no fields extend into the next column. For example, we could have the account description “bleed” into the current-month debit column, which was the case below when Excel initially proposed the column separators.



With some adjusting of the separators, we can see in the display below that the data can now be parsed correctly.

Left half

Data preview

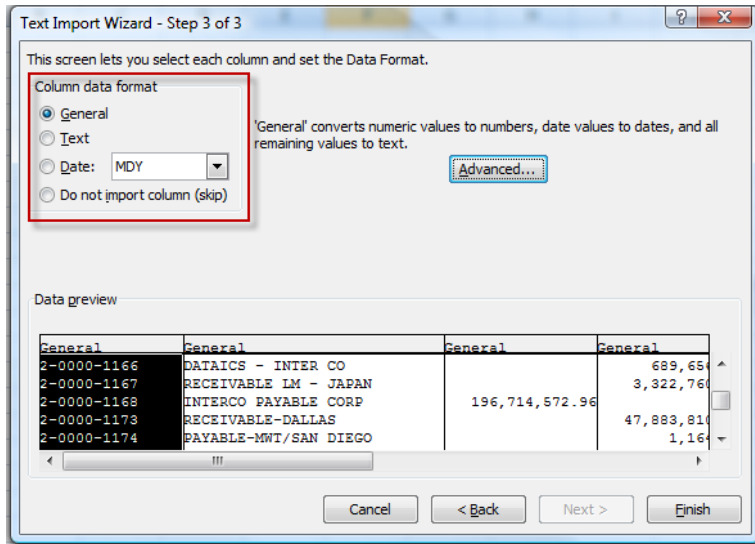
	10	20	30	40	50	60	70
2-0000-1166	DATAICS - INTER CO					689,65	
2-0000-1167	RECEIVABLE IM - JAPAN					3,322,76	
2-0000-1168	INTERCO PAYABLE COOP			196,714,572.96			
2-0000-1173	RECEIVABLE-DALLAS					47,883,81	
2-0000-1174	PAYABLE-MNT/SAN DIEGO					1,16	

Right half

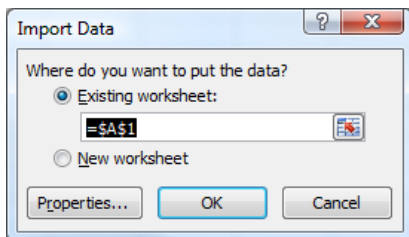
Data preview

	80	90	100	110	120	130	140
656.38							
760.45							
810.00	4,388,702.00		74,618,106.29	74,618,106.29			
164.46			2,911.15	2,911.15			

Once we have our column borders set correctly, we select Next to move on to the third step of the Text Import Wizard. Here in the third step, we can change the data type Excel will utilize when the data are imported. In our example, the General data type will suffice for all our fields. Occasionally, you will get some unexpected results when Excel imports data. One of our client's cost centers started with a single digit, followed by a hyphen and four more characters (e.g. 3-9875 might be for Marketing). When importing this field as General, Excel would convert the cost center value into a date (in reality it is a text string). The date Excel would convert our client's 3-9875 cost center into was March 1, 9875 or 3/1/9875. We can change Excel's default data type in the upper-left corner. We would just click on the column and then select the data type. Excel will force the data to whatever type we specify on import.



Once we have set our data formats for each column or at least reviewed them, we now Finish. Excel will ask you if you want to import the data into the current workbook or into a new workbook as well as where we want the import start, e.g. cell A1 in this example. We just accepted the default below.

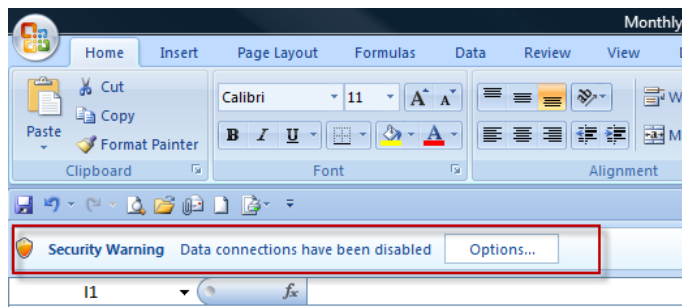


Once we select OK, Excel will then perform the import, which in our case appears as follows:

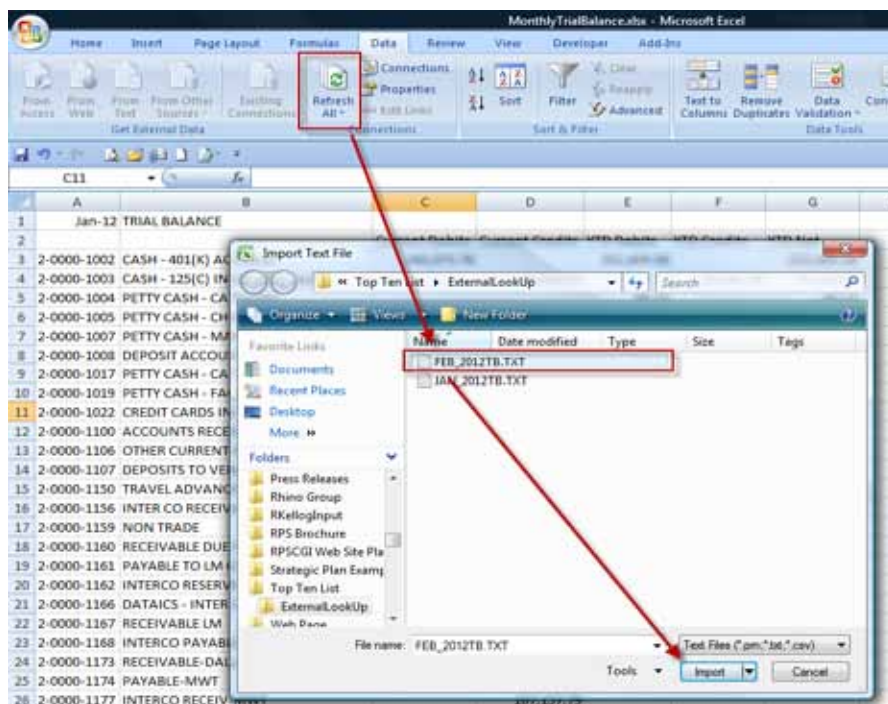
	A	B	C	D	E	F	G
1	Jan 12	TRIAL BALANCE					
2			Current Debits	Current Credits	YTD Debits	YTD Credits	YTD Net
3	2-0000-1002	CASH - 401(K) ACCOUNT	542,073.76		252,604.58		-252,604.58
4	2-0000-1003	CASH - 125(C) INSURANCE ACCT		25,542.99		25,542.99	25,542.99
5	2-0000-1004	PETTY CASH - CATEGORY			48.11		-48.11
6	2-0000-1005	PETTY CASH - CHW/DATA			1,000.00		-1,000.00
7	2-0000-1007	PETTY CASH - MANUF			500		-500
8	2-0000-1008	DEPOSIT ACCOUNT-TREASURY NYC			25,541.00		-25,541.00
9	2-0000-1017	PETTY CASH - CAO			500		-500
10	2-0000-1019	PETTY CASH - FACILITIES			500		-500
11	2-0000-1022	CREDIT CARDS IN-TRANSIT		9,810.92	184,255.00		-184,255.00
12	2-0000-1100	ACCOUNTS RECEIVABLE - TRADE		11,545,343.59			
13	2-0000-1106	OTHER CURRENT ASSETS					
14	2-0000-1107	DEPOSITS TO VENDORS	29,913.36		29,913.36		-29,913.36
15	2-0000-1150	TRAVEL ADVANCES		21,434.43			
16	2-0000-1156	INTER CO RECEIVABLE		529,392.14	306,369.75		-306,369.75
17	2-0000-1159	NON TRADE		59,304.32			
18	2-0000-1160	RECEIVABLE DUE FROM CUST		4,883,648.19			
19	2-0000-1161	PAYABLE TO LM CUST - NVR		97,913.47			
20	2-0000-1162	INTERCO RESERVE		1,004,404.00			
21	2-0000-1166	DATAICS - INTER CO		689,656.38			
22	2-0000-1167	RECEIVABLE LM		1,322,760.45			
23	2-0000-1168	INTERCO PAYABLE CORP				18,106.29	18,106.29
24	2-0000-1173	RECEIVABLE-DALLAS		53,810.00	-388,702.00		-388,702.00
25	2-0000-1174	PAYABLE-MWT		1,164.46		2,911.15	2,911.15
26	2-0000-1177	INTERCO RECEIV-MWT		197,157.79			

At this point, we select Save and close the file. The connection to our text file has now been established within the MonthlyTrialBalance workbook. All the hard work is done relative to establishing the connection and defining the way we want Excel to parse the data each time we refresh our connection to our month trial balance report.

Now assume that a month has past and we need to load the February trial balance report. When we first open the workbook with your new connection we will be warned that our Data connections have been disabled. This is a function of our security settings. We go ahead and select Options to Enable Connections on this content. Since it is our file, we are comfortable with making the connection.



After we have enabled the connection, we go to the data tab, and select Refresh All. We will then be presented with the Import Text File dialog file below (notice we did not have to tell Excel the file type), after which we then select the current-month report and click Import.



Once we click import, Excel wipes out the old trial balance data and imports the new data. We did not have to tell Excel how to parse the flat file nor did we tell it how to force the data types for each column. All that information is saved in the connection that was created when we imported the January trial balance. This is a huge time saver for users that import the same fixed width files on a recurring basis. Our February Trial Balance is displayed below.

	A	B	C	D	E	F	G
1	Feb-12	TRIAL BALANCE					
2			Current Debits	Current Credits	YTD Debits	YTD Credits	YTD Net
3	2-0000-1002	CASH - 401(K) ACCOUNT	133,702.57			214,846.07	214,846.07
4	2-0000-1005	PETTY CASH - CHW/SBLT			1,000.00		-1,000.00
5	2-0000-1007	PETTY CASH - MANUF			500		-500.00
6	2-0000-1008	DEPOSIT ACCOUNT-TREASURY NYC					
7	2-0000-1106	OTHER CURRENT ASSETS					
8	2-0000-1154	IC TRADE REC					
9	2-0000-1156	INTER CO RECEIVABLE	653,361.00		1,091,906.93		-1,091,906.93
10	2-0000-1164	IC NONTRADE REC	238.52		10,411.68		-10,411.68
11	2-0000-1168	INTERCO PAYABLE CORP		2,089,896.09		866,019.75	866,019.75
12	2-0000-1171	INTER CO PAY				1,000.00	1,000.00
13	2-0000-1172	PAYABLE TO MWT		147.42		1,574.78	1,574.78
14	2-0000-1173	RECEIVABLE-DALLAS	792,402.20		262,905.15		262,905.15
15	2-0000-2000	ACCOUNTS PAYABLE -TRADE					
16	2-0000-2010	ACCOUNTS PAYABLE ACCRUAL					
17	2-0000-2017	CLEARING - PCARD		734.68	0.87		-0.87
18	2-0000-2019	INTERCO PAY - SWISS					
19	2-0000-2026	INTERCO PAY - CANADA		26,953.78		26,953.78	26,953.78
20	2-0000-2029	NON TRADE PAY-MWT SINGAPORE	28,793.65				
21	2-0000-2220	GARNISHMENTS W/H		167.50			
22	2-0000-2290	ACCRUED EXPENSES - OTHER		635.62			
23	2-0000-2295	FEDERAL TAXES PAYABLE				362,999.00	362,999.00
24	2-0000-2296	STATE TAXES PAYABLE					
25	2-0000-2300	DEFERRED RENT	62,500.00			8,625,000.00	8,625,000.00
26	2-0000-2752	RECEIVABLE -					

Setting up a connection for a .TXT file is the most basic type of connection we can make in Excel. We can create more complex connections that access data in Access databases, SQL servers, etc. So explore the various connections types.

Tip #5 -- Leveraging Excel's VLOOKUP() function

Background

The often maligned VLOOKUP() function really has many uses for accountants and analysts. From our experience, VLOOKUP() really adds value when we need to add dimensions before performing a specific analysis. For example, as part of larger projects clients will provide a list of inventory values, by material number, and ask us to segregate the inventory values by type (e.g. raw materials, semi-finished subassemblies and finished goods). In addition to the inventory listing by material number, we are provided another list with inventory types by material number. Using both lists, we can perform a VLOOKUP() and add the material types to our inventory value listing. This would be a fairly simple process if the data were in a database by just performing a JOIN between the two tables on the material numbers. But many times, it is faster and easier to perform a VLOOKUP() in Excel.

Application

The VLOOKUP() function takes four parameters that are as follows:

VLOOKUP(*lookup_value*, *lookup_array*, *return_column_index*, *range_lookup*)

1. *Lookup_value* – This is the value to be looked up (or matched) in the first column of our array specified in the second parameter (*lookup_array*). Users need to be aware of what data type is being provided as a *lookup_value* as well the data type in *lookup_array*. For example, assume we are searching on cost centers. A cost center of 58342 could be a text string in our *lookup_array* and an actual value as our *lookup_value*. When this occurs, VLOOKUP() will not be able to find a matching value, because the text string "58342" is not the same as the value 58,342; consequently, VLOOKUP() will return NA# indicating no matching value exists. Caveat #1: make sure that both the *lookup_value* and first column of *lookup_array* have the same data type. Tip: We can use the ISTEXT() function to determine if a value is a text string (if ISTEXT() returns TRUE, then the reference is text, while FALSE indicates a value or some other data type).
2. *Lookup_array* – We need to provide the address of a two-dimensional array that in column one contains the values to be searched. We can provide either a range of cells, e.g. A12:D92, or a Defined Name range (see Tip #2 for more information on named ranges). In either case, the *Lookup_array* can be on the current worksheet, another worksheet in the same workbook or another worksheet in a different workbook.
3. *Return_column_index* – This is the column index for the values we want returned by VLOOKUP(). If we provide range A12:D92 as our *Lookup_array* and "4" as our *return_column_index* number, then whenever a match is made on our *lookup_value* the value in the fourth column (or column D), at the row of our match, is returned by VLOOKUP().
4. *Range_Lookup* – The value of the fourth parameter can be either FALSE, TRUE or omitted. If it is omitted, it is assumed to be TRUE. This setting tells the VLOOKUP()

function to either find an exact match (the FALSE setting), or the closest match if an exact match does not exist (the TRUE setting). If the setting is FALSE, and an exact match is not found, the VLOOKUP() returns NA#. Later, we will discuss how to use this behavior to our advantage. If there are multiple correct values, then VLOOKUP() returns the first occurrence. CAVEAT #2: Be cautious of multiple occurrences of *lookup_value* in the *lookup_array*.

We are going to use our inventory categorization example, with material numbers and values from a hypothetical month end, and merge the material descriptions, unit standard costs, material types (raw material, sub-assembly and finished goods), as well as calculate the quantity on hand. This is a good example of merging other dimensional data (standard cost, material type and description) that would enable other more specific analysis.

Our monthly inventory report only includes the material numbers and the extended values. The top of the report appears as follows:

	A	B	C	D	E	F	G	H
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								

ME July 2012 Inventory Report					
Material #	Inventory Value	Inventory Type	Description	Standard Cost	Quantity On Hand
520468-006	\$8,239.52				
236893-099	\$904.98				
348208-074	\$7,167.78				
880753-030	\$486.26				
989146-040	\$2,754.07				
615408-092	\$436.77				
179591-032	\$88.25				
646231-066	\$929.28				
182787-026	\$4,311.00				
725340-095	\$586.32				
046890-091	\$791.76				

The orange-shaded section is the data that we want to either VLOOKUP() or calculate using data returned by VLOOKUP(). We have also been provided with the material master, which appears as follows:

	A	B	C	D	E	F
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						

Material Master				
Material #	Material Desc	Inventory Type	Standard Cost	
010537-031	Material Item: 010537	Raw Material	\$0.00	
031395-082	Material Item: 031395	Raw Material	\$0.18	
046890-091	Material Item: 046890	Raw Material	\$0.89	
055066-022	Material Item: 055066	Raw Material	\$0.04	
059860-076	Material Item: 059860	Raw Material	\$0.38	
179591-032	Material Item: 179591	Raw Material	\$0.01	
223569-059	Material Item: 223569	Raw Material	\$0.09	
236893-099	Material Item: 236893	Raw Material	\$0.25	
257071-081	Material Item: 257071	Raw Material	\$0.21	
437772-060	Material Item: 437772	Raw Material	\$0.12	
469881-002	Material Item: 469881	Raw Material	\$2.18	

The material master contains the material number, description, type and standard cost. Both worksheets are in the same workbook. We will create VLOOKUP() formulas for these four master data fields and calculate the quantity on hand. The first item to be retrieved will be the inventory type. The *lookup_array* will begin at B7 (column B contains our lookup

values) and extend to the last row column E, which is row 49 in our Material Master file. Our VLOOKUP() to retrieve the inventory type is as follows:

=VLOOKUP(C5,Material_Master!\$B\$7:\$E\$51,3,FALSE)

In the worksheet, it will appear as follows:

The screenshot shows the Excel formula bar with the formula `=VLOOKUP(C5,Material_Master!B7:E51,3,FALSE)`. Below the formula bar, a red arrow points from cell E5 to the 'Inventory Type' column of the 'ME July 2012 Inventory Report' table. The table has the following data:

Material #	Inventory Value	Inventory Type	Description	Standard Cost	Quantity On Hand
520468-006	\$8,239.52	Sub-Assembly			
236893-099	\$904.98				
348208-074	\$7,167.78				
880753-030	\$486.26				
989146-040	\$2,754.07				

We can see that VLOOKUP() returned the correct inventory type, corresponding to the lookup_value (column C material numbers), which is in the 3rd row of our lookup_array (Material_Master!\$B\$7:\$E\$51) on the Material_Master worksheet. Now we just need to use the same formula to retrieve the description and standard cost. The VLOOKUP() formulas for each column, at row 5, are as follows:

E5 = VLOOKUP(C5,Material_Master!\$B\$7:\$E\$51,3,FALSE) <-Inventory type
 F5 = VLOOKUP(C5,Material_Master!\$B\$7:\$E\$51,2,FALSE) <-Description
 G5 = VLOOKUP(C5,Material_Master!\$B\$7:\$E\$51,4,FALSE) <-Standard cost

Once we have the standard cost per unit then we will calculate the quantity on hand. Here is the top section of our inventory report with all the columns completed.

The screenshot shows the Excel formula bar with the formula `=D5/G5`. Below the formula bar, the 'ME July 2012 Inventory Report' table is shown with all columns completed:

Material #	Inventory Value	Inventory Type	Description	Standard Cost	Quantity On Hand
520468-006	\$8,239.52	Sub-Assembly	Material Item: 520468	\$54.21	152
236893-099	\$904.98	Raw Material	Material Item: 236893	\$0.25	3,614
348208-074	\$7,167.78	Sub-Assembly	Material Item: 348208	\$13.45	533
880753-030	\$486.26	Raw Material	Material Item: 880753	\$0.07	6,657

Since the goal was to provide a summary by inventory type, we just need to sort the inventory report by inventory type and material number and subtotal the report. The sorted and subtotaled report will appear as follows:

=VLOOKUP(C14,Material_Master!\$B\$7:\$E\$51,4,FALSE)					
C	D	E	F	G	H

ME July 2012 Inventory Report					
Material #	Inventory Value	Inventory Type	Description	Standard Cost	Quantity On Hand
074925-062	\$5,076.65	Finished Good	Material Item: 074925	\$6.80	747
351096-028	\$7,317.12	Finished Good	Material Item: 351096	\$66.52	110
491680-093	\$5,932.91	Finished Good	Material Item: 491680	\$11.91	498
648400-013	\$104.16	Finished Good	Material Item: 648400	\$0.11	937
696527-046	\$4,785.69	Finished Good	Material Item: 696527	\$22.36	214
965499-004	\$5,863.67	Finished Good	Material Item: 965499	\$177.69	33
	\$29,080.20	Finished Good Total			2,539
010537-031	\$2.81	Raw Material	Material Item: 010537	\$0.00	7,075
031395-082	\$897.25	Raw Material	Material Item: 031395	\$0.18	5,067
046890-091	\$791.76	Raw Material	Material Item: 046890	\$0.89	892

A great tip on making subtotal reports more readable is to add a border and color to the subtotals. This is done using conditional formatting. The formula we need to apply, after selecting the entire report, searches column E for any description that contains, as the right most five characters, the word "TOTAL". All rows that meet this criteria are highlighted with green and receive a top border. The expanded and collapsed subtotal reports appear as follows:

Expanded

=VLOOKUP(C18,Material_Master!\$B\$7:\$E\$51,2,FALSE)					
C	D	E	F	G	H

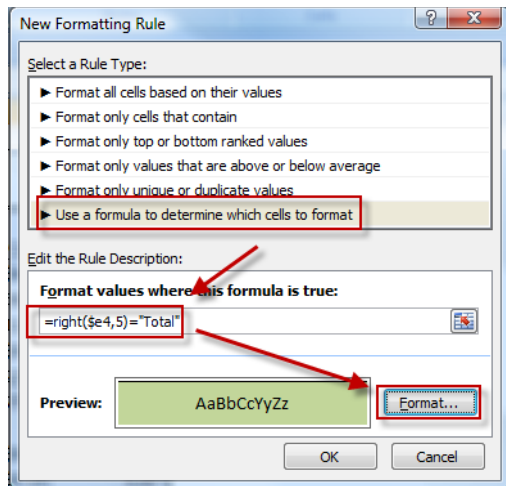
ME July 2012 Inventory Report					
Material #	Inventory Value	Inventory Type	Description	Standard Cost	Quantity On Hand
074925-062	\$5,076.65	Finished Good	Material Item: 074925	\$6.80	747
351096-028	\$7,317.12	Finished Good	Material Item: 351096	\$66.52	110
491680-093	\$5,932.91	Finished Good	Material Item: 491680	\$11.91	498
648400-013	\$104.16	Finished Good	Material Item: 648400	\$0.11	937
696527-046	\$4,785.69	Finished Good	Material Item: 696527	\$22.36	214
965499-004	\$5,863.67	Finished Good	Material Item: 965499	\$177.69	33
	\$29,080.20	Finished Good Total			2,539
010537-031	\$2.81	Raw Material	Material Item: 010537	\$0.00	7,075

Collapsed

C	D	E	F	G	H

ME July 2012 Inventory Report					
Material #	Inventory Value	Inventory Type	Description	Standard Cost	Quantity On Hand
	\$29,080.20	Finished Good Total			2,539
	\$12,327.18	Raw Material Total			96,929
	\$91,131.08	Sub-Assembly Total			8,352
	\$132,538.46	Grand Total			107,820

To set the conditional formatting, navigate to Home menu tab and within the styles group select Conditional Formatting. You will want to first highlight the entire area to be applied with the conditional formatting with the column headers, and then use the formula and settings below to highlight the rows as shown in the example above.



Note that the conditional formatting starts at the column header row. This is because we selected the column header row in our range before selecting the Conditional Formatting option. The "right(\$e4,5)" means that the conditional format function will search for the word "Total" in column E only, beginning with row 4, and continuing down through all the selected rows. The ",5)" means that the Conditional Formatting will grab the last 5 characters, which will be "Total" for the Excel generated subtotal and total lines.

A more advanced use for VLOOKUP() -- Controlling for NA#

We nearly always use the "FALSE" setting for the fourth VLOOKUP() parameter—*range_lookup*. Remember this is the parameter that determines if Excel must find an exact match (=FALSE), or the closest match if an exact match does not exist (either omitted or =TRUE). Knowing that Excel will return a NA# error if the fourth parameter is set to FALSE allows us to leverage this behavior to extend the usage of VLOOKUP(). This is especially important if we are going to try and perform any calculations with the numbers that are returned from our VLOOKUP() function, because returning an NA# to a SUM() expression, is going to return an NA# from the SUM() as well; consequently we have to allow for the potential that VLOOKUP() may return a NA# rather than a value.

Excel has a number of IS() functions that provide information about the condition or state of a cell or value. With the VLOOKUP() function we frequently use the ISNA() function. If an ISNA() function returns TRUE, then whatever value, result, or reference that exists between the parenthesis is returning a NA#. To allow for potential NA# values, we incorporate an ISNA() check into our VLOOKUP() function. In the original VLOOKUP(), we used the following formula to return the inventory type:

```
=VLOOKUP(C5,Material_Master!$B$7:$E$51,3,FALSE)
```

What if we know that we might have material numbers on the Inventory Report that are not referenced in our Material Master list. In these cases, we want the formula to return "Not Found" when our VLOOKUP() cannot find an exact match for our VLOOKUP(). To accomplish this verification, the above formula would be changed to the following:

```
=IF(ISNA(VLOOKUP(C5,Material_Master!$B$7:$E$51,3,FALSE)),"Not Found",  
    VLOOKUP(C5,Material_Master!$B$7:$E$51,3,FALSE))
```

The above formula will return "Not Found" to the inventory type column if there is no exact match to the *lookup_value*. The green font represents the VLOOKUP() needed to check if the value exists, the VLOOKUP() in orange font actually returns the inventory type if the material number exists in the Material Master table. If we are doing calculations with the returned value we will want to return the value zero.

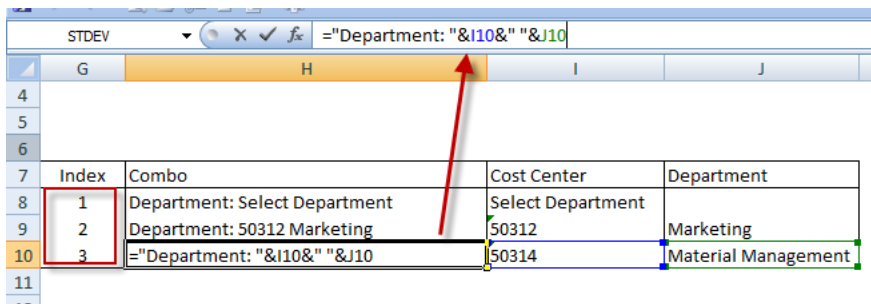
Other VLOOKUP() Tips

Sometimes we concatenate two fields together to derive a unique key and use it as our *lookup_value*. For example, you may have one file with customer revenue dollars by customer number and material number and you may just want to analyze certain products by selected customers. You can create new *lookup_values* by concatenating the customer number with the material number, using some delimiter between the two text strings, such as an "*". So if we had customer numbers in column B and material numbers in column C, we could use =B3&"*"&C3 at cell D3 to create a new key. Then we would use this new value as our *lookup_value*.

Another caveat regarding exact matching on text strings pertains to blank spaces. Periodically, we see clients get really frustrated when using VLOOKUP() because they have text strings that they know exist in their *lookup_array*, but Excel keeps returning NA#. Typically this is caused by a trailing blank space in either the *lookup_value* or the *lookup_array* that users cannot easily see given that a trailing space is invisible when looking at the values on a computer screen. To eliminate the potential for trailing spaces, we use the TRIM() function to remove both leading or trailing spaces. Use TRIM() both on the *lookup_value* side as well as on the *lookup_array* side to prevent this issue.

Ó 2010 Resource Planning Solutions

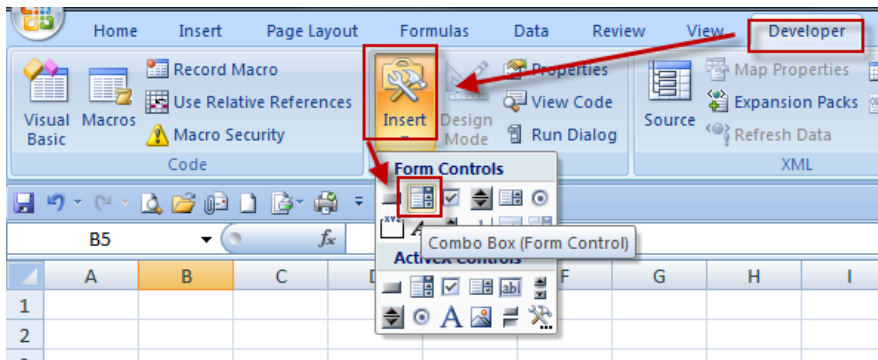
At the outset of a budget exercise, the user must identify the department number for their budget worksheet. For both the cost center and account number identification, we have used Excel's Combo Box Form Control. The Combo Box does not require Visual Basic to provide some valuable functionality. We created a list of three departments on the MasterData worksheet, the first being the default "Select Department" option. So there are only two cost centers set up as you can see below.



Index	Combo	Cost Center	Department
1	Department: Select Department	Select Department	
2	Department: 50312 Marketing	50312	Marketing
3	= "Department: "&I10&" "&J10	50314	Material Management

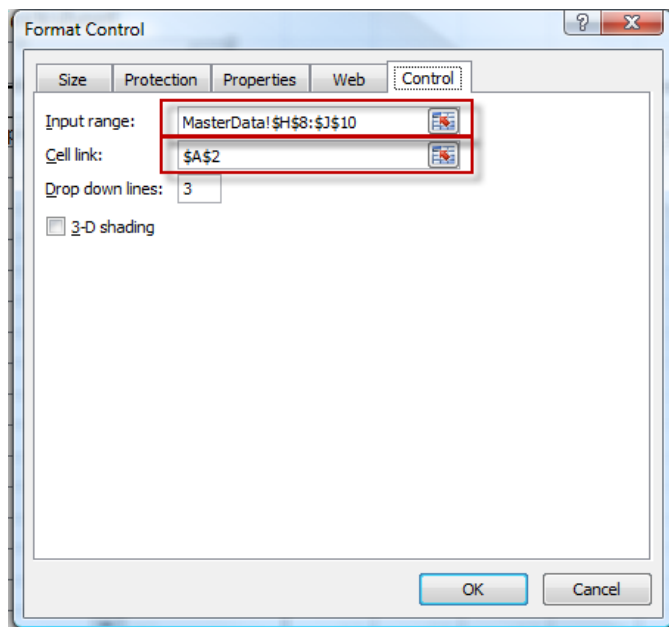
Columns I and J contain the cost centers and department descriptions, respectively, that we downloaded from the accounting system. We want our Combo Box to read from column H which is a simple formula to concatenate the cost center and department together. There is one other very important piece to this matrix, which is the index number at column G. This was hard coded from 1 to 3. One of the features of the combo box is that it will return the index number for a selected item. This means that if a user selects the 3rd item on a Combo Box list, then Excel will store the number 3, or the index for the user selection, into a cell of our choosing.

By referencing the H8:H10 cell range in our combo box, all departments will be selectable on our combo box. To create a combo box, we will need to navigate to Excel's Developer menu tab, select Insert and then click on the Form Control Combo Box as shown below (The top section contains the Form Controls, which is what we want, while the lower contains ActiveX Controls. The ActiveX Controls require more programming, but ultimately are much more capable than the Form Controls.)



Once we click on the Combo Box icon, we can then drag and size our object. Initially, there is no description in for our Combo Box. The description will appear once we define the

Input Range. With the Input Range defined, the Combo Box will display our first item from the Input Range list. To define the Input Range, we right click the Combo Box on our worksheet, and select Format Control..., which will bring up the Format Control dialog below.



On the Control tab, we added our H8:H10 range from the MasterData worksheet. This will contain our two cost centers and the "Select Department" default option, which totals three choices. Note the Cell link, with an address at \$A\$2. When a user selects a cost center, Excel returns the index number for that choice (it will be between 1 and 3 for number of cost centers in our H8:H10 range) and stores it at our specified location--\$A\$2. This is important, because we will use the index later in a VLOOKUP() function to help us retrieve the cost center. We right click the Format Control and move it just beneath the title to our report.

	A	B	C	D	E
1		2012 Budget Worksheet			
2	1	Department: Select Department ▼			
3					
4		Acct # / Acct Description		Acct Group	

Note that cell A2 contains 1. This is because our "Select Department" is the first item on our range, or H8 on MasterData. This number will change each time a user selects the drop down and changes the cost center selection (A2 will become 2 when the second item is chosen and so on).

#2 Adding general ledger account Combo Boxes

We do the same for the account number and account descriptions on the MasterData worksheet by combining the account and description into one column and then setting the

Input Range to reference that column on MasterData, which is B8 to B33. Then we position the Combo Box and link it to the adjacent cell starting at A5. Next, we copy and paste 19 more Combo Boxes for rows 6 through 24, setting the linked cell in each row to the column A cell adjacent to the Combo box (e.g. the Combo Box at row 19 is linked to cell \$A\$19).

The MasterData table for general ledger accounts, including the concatenated column referenced by our Combo Boxes, at column B, appears as follows:

	A	B	C	D	E
5					
6		Account Listing			
7	Index	Combo	Acct#	Desc	Acct Group
8	1	Assign Code			TBD
9	2	0500 EXEMPT LABOR	0500	EXEMPT LABOR	WAGES & FRNG
10	3	0510 NON EXEMPT LABOR	0510	NON EXEMPT LABOR	WAGES & FRNG
11	4	0520 VACATION	0520	VACATION	WAGES & FRNG
12	5	0560 COMMISSIONS	0560	COMMISSIONS	WAGES & FRNG
13	6	0595 FRINGE BENEFIT	0595	FRINGE BENEFIT	WAGES & FRNG
14	7	0600 EMPLOYMENT ADVERTISING	0600	EMPLOYMENT ADVERTISING	NEW HIRE
15	8	5001 OPERATIONAL SUPPLIES	5001	OPERATIONAL SUPPLIES	SUPPLIES & MATERIAL
16	9	5003 FIELD SUPPLIES	5003	FIELD SUPPLIES	SUPPLIES & MATERIAL
17	10	5006 FLOOR STOCK	5006	FLOOR STOCK	SUPPLIES & MATERIAL

#3 Adding the general ledger account group via VLOOKUP() on index

Note on the Account Listing above, that adjacent to the account number and description, we included an account group (Acct Group). On our BudgetSheet, we also pull in the account group ("WAGES & FRNG" in the example below) at E5.

		Column D	Column E
4		Acct # / Acct Description	Acct Group
5	2	0500 EXEMPT LABOR	WAGES & FRNG
6	1	Assign Code	TBD
7	1	Assign Code	TBD

BudgetSheet

This is done with a VLOOKUP(), referencing the index at row 5 (\$A\$5), which is the value 2 in our example above, and then returning the Acct Group—WAGES & FRNG—from MasterData. The accounts table on MasterData appears as follows:

	A	B	C	D	E
6	Account Listing				
7	Index	Combo	Acct#	Desc	Acct Group
8	1	Assign Code			TBD
9	2	0500 EXEMPT LABOR	0500	EXEMPT LABOR	WAGES & FRNG
10	3	0510 NON EXEMPT LABOR	0510	NON EXEMPT LABOR	WAGES & FRNG
11	4	0520 VACATION	0520	VACATION	WAGES & FRNG
12	5	0560 COMMISSIONS	0560	COMMISSIONS	WAGES & FRNG

This is the formula on BudgetSheet (at cell E5) that returns "WAGES & FRNG"

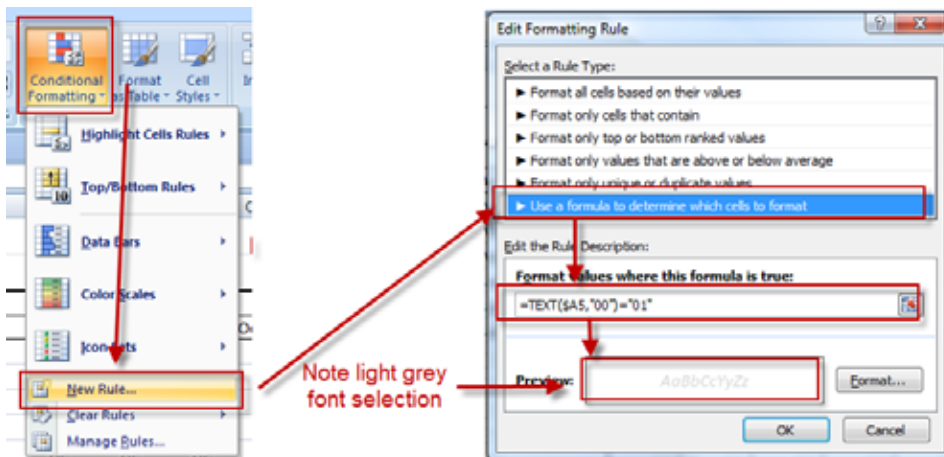
`=VLOOKUP(A5,MasterData!A8:E33,5,FALSE)`

#4 Using conditional formatting to help users identify active rows

We can see from the example below that when a user selects an account, that the linked cell for that row will change from 1 to some number greater than one. We are using a conditional format to change the font for rows 5 through 24 from a light grey color to the default black color. This is a "nice to have," but it provides another level of visual clues to the user that information is required in those cells.

2012 Budget Worksheet															
2	Department: 0000 Marketing														
			(\$000)												
	Acct # / Acct Description	Acct Group	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	2012
3	0000 EXEMPT LABOR	WAGES & FRNG	15.0	20.0	25.0	30.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	90.0
4	0000 FRINGE BENEFIT	WAGES & FRNG	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0000 VACATION	WAGES & FRNG	5.0	40.0	50.0	60.0	7.0	6.0	0.0	0.0	0.0	0.0	0.0	0.0	168.0
6	0000 OPERATIONAL SUPPLIES	SUPPLIES & MATERIA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0000 FIELD SUPPLIES	SUPPLIES & MATERIA	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	60.0
8	0000 FLOOR STOCK	SUPPLIES & MATERIA	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	Assign Code		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	Assign Code		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	Assign Code		0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

To set a conditional format, select the entire range of cells to be formatted (A5:T24 in this example). Then from the Home menu tab, select Conditional Formatting from the Styles group. From the drop down, select New Rule. From the Edit Formatting Rule select the Formula option. The formula takes the value at \$A5 and converts to a text string of two characters with a preceding 0 if the value is less than 10. It then checks to see if that is equal to "01". If \$A5 is equal to "01" then the light grey font is applied.



The difficult part to creating Conditional Formats is specifying the starting cell correctly, which is \$A5. Remember that before we selected the Conditional Formatting function, we had selected cells A5:T24 as the range to apply the Conditional Format. By identifying the column as an absolute reference, e.g. "\$A" and the row reference as relative, e.g. "5", then the Conditional format will only evaluate column A, from row 5 through row 24 to determine whether or not to change the font.

#5 Summarize the total budget and compare it to prior-year spending

On the right-hand side of our budget application you will see that we are totaling the user input in column R; so the total for row 5 is calculated at cell R5 using SUM(F5:Q5). In the next column, you will notice a Prior Year column with a number of values and the Delta or variance between the current estimate and the prior-year actual spend (note below we have hidden most of the months).

	B	C	D	E	F	Q	R	S	T
1	2012 Budget Worksheet								
2	Department: 50312 Marketing								
3									
4									
5									
6									
7									
8									
9									
10									
11									

The 2011Actuals worksheet contains the prior-year actuals, by account for each cost center. We have also added a column that combines the cost center and account to create a key field, the two fields being delimited with an "*", that we will use to search via a VLOOKUP() function from the BudgetSheet worksheet (creating search keys for use with VLOOKUP() is discussed in Tip #4).

	A	B	C	D
7	Key Field	Cost Center #	Account #	2011 Spend
8	50312*0500	50312	0500	257.5
9	50312*0510	50312	0510	30.3
10	50312*0520	50312	0520	226
11	50312*0560	50312	0560	174
12	50312*0600	50312	0600	134.2
13	50312*5001	50312	5001	200.1
14	50312*5003	50312	5003	113.1
15	50312*5010	50312	5010	29.3
16	50312*6004	50312	6004	93.7
17	50312*6005	50312	6005	247.2
18	50312*6100	50312	6100	179

By creating the key on the 2011Actuals worksheet, we can now use this key on the BudgetSheet to return, from 2011Actuals, prior-year actuals and calculate both a dollar and % variance. Here is how it works.

Returning to the BudgetSheet and expanding the range, we can now see that there is a similar key in column V, where we are concatenating the cost center and account into a key field. Remember that since we are using Combo Boxes, we have to use VLOOKUP() with the indexes in column A to determine the actual values to be concatenated. So there is one VLOOKUP() for the cost center and one for account number. Both VLOOKUP()s exist in column V.

The screenshot shows the 2012 Budget Worksheet with a table of budget items. The formula in cell V5 is: `=IF(A5<>"",VLOOKUP(SA$2:MasterData!$G$8:$J$10,3,FALSE,"")&"*"&VLOOKUP(A5:MasterData!A8:C33,3,FALSE,""),"NA")`. Red arrows indicate that the first VLOOKUP fetches the cost center number and the second VLOOKUP fetches the account number.

Acct # / Acct Description	Acct Group	Jan	Dec	2012	Prior Yr	Delta	CC-Acct	Delta %
0000 EMERIT LADOR	WAGES & FRING	25.0	0.0	90.0	297.5	147.5	50312*0500	65%
0000 FRINGE BENEFIT	WAGES & FRING	0.0	0.0	0.0	0.0	0.0	50312*0595	0%
0020 VACATION	WAGES & FRING	5.0	0.0	168.0	226.0	58.0	50312*0520	26%
5000 OPERATIONAL SUPPLIES	SUPPLIES & MATERIA	0.0	0.0	0.0	200.1	200.1	50312*5001	100%
5003 FIELD SUPPLIES	SUPPLIES & MATERIA	5.0	5.0	60.0	113.1	53.1	50312*5003	47%

Now that column V contains our search key (cost center + "*" + account number), we can use it in column S to search the 2011 Actuals worksheet for the Prior Year spend for comparison to the current 2012 budget input.

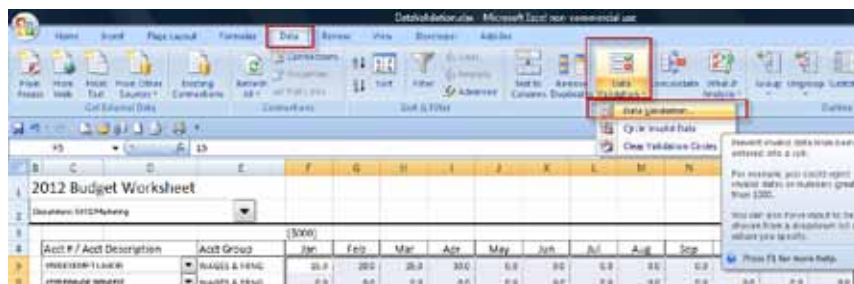
The formulas used to retrieve the prior-year actuals are in column S. Again, column S contains a VLOOKUP() using the key in column V, to search the 2011Actuals worksheet. The formula in column S is not as complex as it might appear. As explained in the section on VLOOKUP(), we are using one of Excel's IS() functions to ensure that there is a value that matches the current cost center and account on the 2011Actuals worksheet. [If no match exists—NA# is returned, then the user may be using that account for the first time for their department.] We are using ISNA(), which returns TRUE if the value between the parentheses returns NA#. The way that the VLOOKUP() has been defined, if Excel cannot find an exact match for the key, then the VLOOKUP() returns NA#. The IF() statement that surrounds the ISNA() function returns 0.0 if VLOOKUP() returns NA#; otherwise, without the IF() and ISNA(), Excel could return NA# into column S, which would create other problems. See the Tip #4 on VLOOKUP() for more information. The formula for column S appears as follows:

S5		=IF(V5<>"NA",IF(ISNA(VLOOKUP(V5,"2011Actuals"!\$A\$8:\$D\$46,4,FALSE)),0,VLOOKUP(V5,"2011Actuals"!\$A\$8:\$D\$46,4,FALSE)),0)																					
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
2012 Budget Worksheet																							
Department: 50312 Marketing																							
(\$000)																							
Acct # / Acct Description		Acct Group	Jan	Dec	2012	Prior Yr	Delta	CC & Acct#		Delta %													
0500 EXEMPT LABOR		WAGES & FRNG	15.0	0.0	90.0	257.5	167.5	50312*0500		65%													
0595 FRINGE BENEFIT		WAGES & FRNG	0.0	0.0	0.0	0.0	0.0	50312*0595		0%													
0620 VACATION		WAGES & FRNG	5.0	0.0	168.0	226.0	58.0	50312*0620		26%													
5001 OPERATIONAL SUPPLIES		SUPPLIES & MATERIA	0.0	0.0	0.0	200.1	200.1	50312*5001		100%													
5003 FIELD SUPPLIES		SUPPLIES & MATERIA	5.0	5.0	60.0	113.1	53.1	50312*5003		47%													
5006 FLOOR STOCK		SUPPLIES & MATERIA	0.0	0.0	0.0	0.0	0.0	50312*5006		0%													
Assign Code		100	0.0	0.0	0.0	0.0	0.0	NA		0%													

#6 Using Data Validation to encourage user compliance

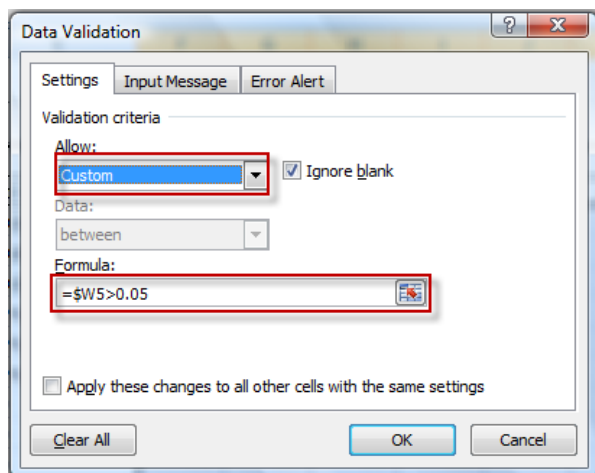
Many times, budget exercises are based on spending from a prior period. In this instance, our hypothetical direction is for budget managers to plan on spending 5% less in 2012 than the prior-year spend. Since we now have their total 2012 spend in column R, the prior year in column S, we can easily calculate the prior year variance in column W (the formula for row 5 is “=IF(S5<>0,(S5-R5)/S5,0)”). Now we want to set up a Data Validation rule that will send a message to the budget manager if their 2012 spend input exceeds 95% of the prior-year spending.

Excel's Data Validation function will permit us to evaluate whether any change made to the cells for January through December, for any row, causes the 2012 spend plan to exceed 95% of the prior-year spend (or the value of column W to exceed 95%). To apply Data Validation to a specific cell, we first need to select the cells that require Data Validation. In this instance, we want all the 2012 budget input cells to be validated, which means cells F5:Q24 are highlighted. Once the cells are highlighted, from the Data menu tab, select Data Validation, from the Tools group, and then select Data Validation...

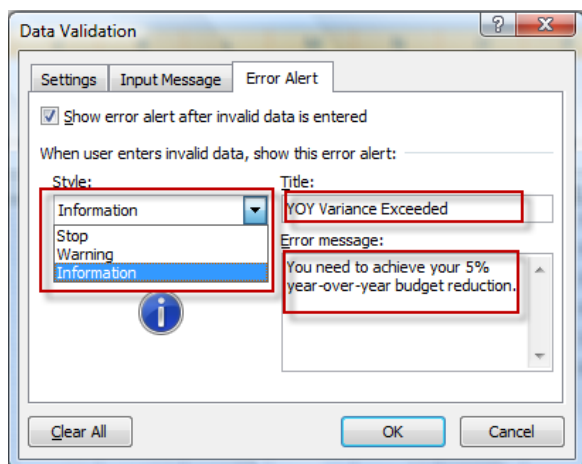


There are three tabs on the Data Validation dialog, which include Settings, Input Message and Error Alert. Beginning with the Settings tab, we change the validation criteria to Custom, because we want to create our own formula. After selecting Custom validation criteria, we can now input our formula (remember that before we opened the Data Validation dialog, we selected all the 2012 input cells—F5:Q24), which is “=\$W5>.05”. Remember that we created a formula at column W that reflects the year-over-year growth rate, positive numbers reflecting less spend than the prior year. By using an absolute reference for the column--\$W—and a relative reference on the row—5—Excel will duplicate

the Data Validation, changing the row number but leaving the column reference the same for our entire range (F5:Q24). So the Data Validation formula at R6 will be “=\$W6>.05” and so on and so forth to cell Q24, which will be “=\$W24>.05”.



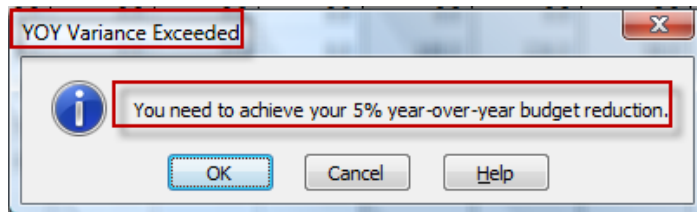
Now that we have the settings established, we will move on to the Error Alert (we are not going to create any Input Message for this application). There are three forms of Error Alerts—Stop, Warning and Information. Given that we want to encourage budget managers to remain within 95% of their prior-year spend rather than prevent them from doing so, we are going to use the Informational message Style. You can see that we have created some simple messages to convey to the user their need to stay within 95% of prior-year spending. Users just need to click OK and they can proceed to budget whatever dollars are required. Once we have input the message, we can now select OK to apply Data Validation.



In the example below, the budget manager has input their 2012 spend plan through September, and has added approximately \$225.0K, which is about 87% to prior-year spending (1 – Delta %=87%) of \$257.5K. No message has been issued.

	B	C	D	E	N	O	P	Q	R	S	T	U	V	W
1	2012 Budget Worksheet													
2	Department: 50312 Marketing													
3														
4	Acct # / Acct Description	Acct Group		Sep	Oct	Nov	Dec	2012	Prior Yr	Delta	CC & Acct#		Delta %	
5	6500 EXEMPT LABOR	WAGES & FRING		40.0	0.0	0.0	0.0	225.0	257.5	32.5	50312*0500		13%	

If the user adds \$25k more to cell O5, they will have forecast to spend more than 95% of their 2011 baseline. Adding \$25K more to October at O5, we now receive the following message from Excel:



The budget manager has two options at this point. Option one is to select Cancel, which will revert the value at O5 to the previous value or select OK which will leave the \$25K unchanged, and the budget manager will have a variance to the published budget guidelines. Adding additional spend to November and December will produce the same warning with the same options—Cancel or OK.

That is about it for this example of Data Validation and user input controls. There are many more things that you can do with the Data Validation function, including preventing users from providing incorrect values. All of this is possible without having to use Visual Basic.

Tip #7 -- Setting up simple user controls

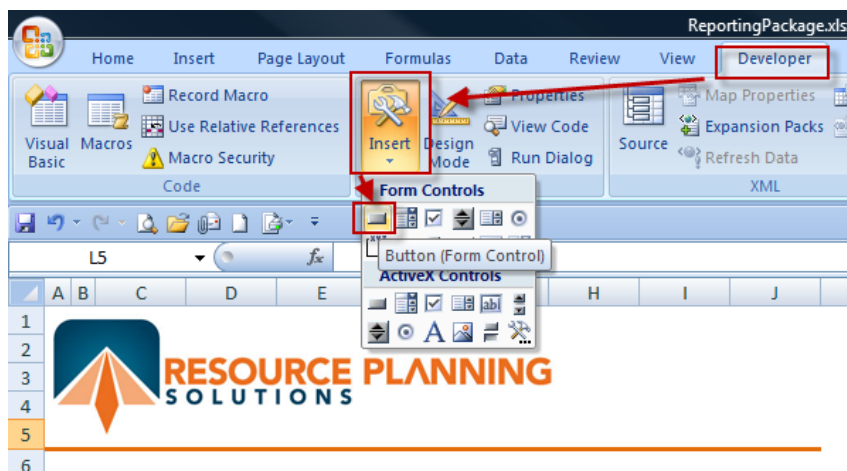
Background

This is a good point to take a look at what is possible with some simple Visual Basic (VB) programming. Users that want to take their Excel skills to the next level, need to get learn basic VB programming. With VB code, spreadsheets applications can be made vastly more capable. Our user would prefer to just click a button on a worksheet and have all their reports printed with one click rather than having to navigate from worksheet to worksheet, printing their reports. Regardless, this Tip is meant as a very basic VB primer. There are many great books, of varying skill level, on this subject, to learn more about VB. Just go to Amazon and search away.

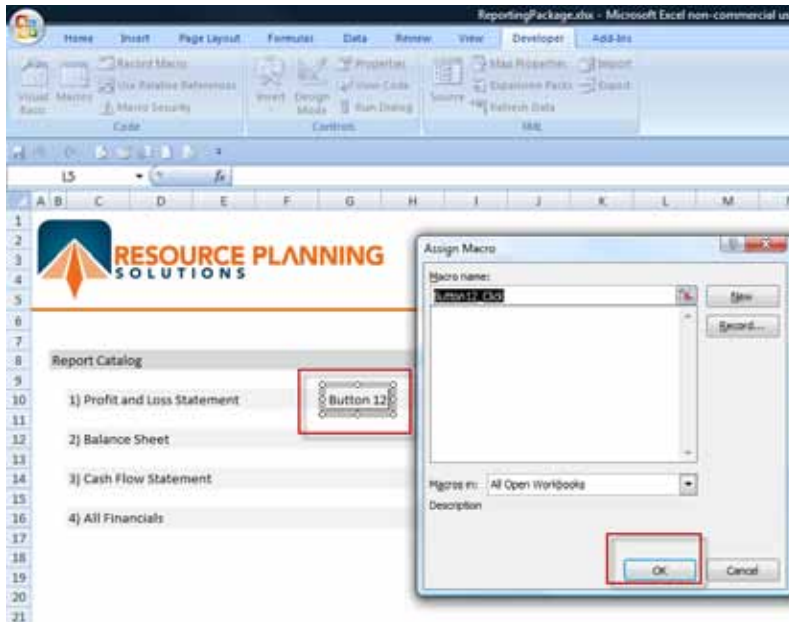
Application

For this example, we are going to implement a simple reporting application that contains three financial reports. There will be four worksheets total, one for each report and one for our user menu. The user menu will contain two buttons for each report. The first button will just navigate to the worksheet or report. The second button will print the report. We will also create a button that prints all the reports with one click.

The worksheets have already been formatted, so the next step will be to create the Button Form Controls, that users will select to run the various macros. From the Developer menu tab, we need to select the Insert option from the Controls group as shown below and click the Button Form Control. Excel will display a "+" which will allow us to drag and draw the object to the desired size.



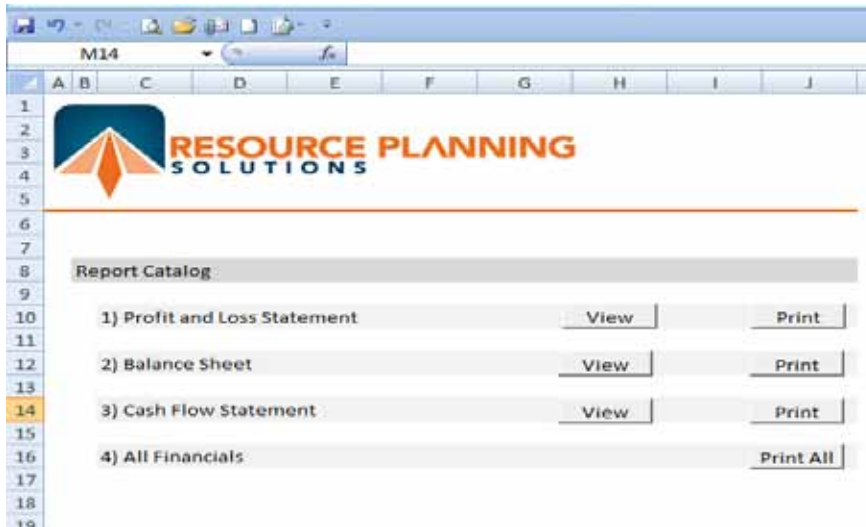
Once we have drawn our Button, Excel will then generate the "Assign Macro" dialog shown below asking us to assign a macro. Since we have not created any Button macros yet, ignore this request for now by selecting "OK". We will assign the macros in a minute.



At this point, we have drawn the button and are about to select “OK” which will produce the Button, with no assigned macro. After selecting “OK”, we copy and paste the Button six times, since we will need a total of seven buttons, as follows: 1) view profit and loss; 2) view balance sheet; 3) view cash flow statement; 4) print profit and loss; 5) print balance sheet; 6) print cash flow statement; and 7) print all three financial statements. Once the seven Buttons have been pasted, then change the button titles by holding down the control key while hovering over the Button to be changed and right click. Then release the control key and right click the button a second time, and you should be able to change the title of the Button.

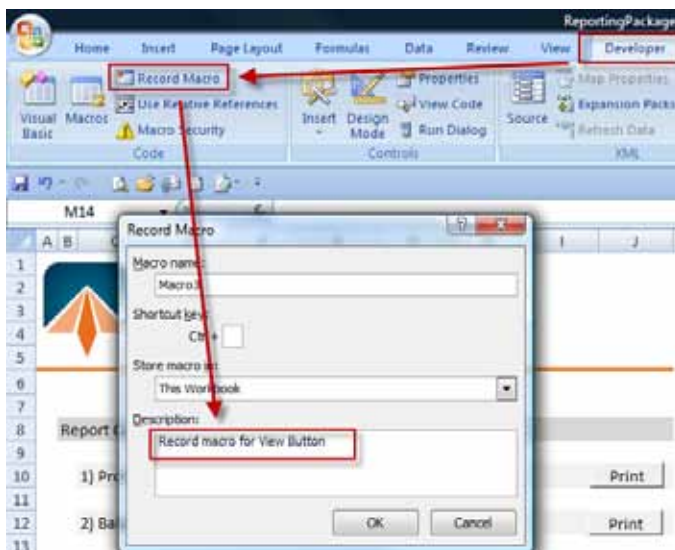


Once you have edited each of the seven buttons, align the buttons as shown below.

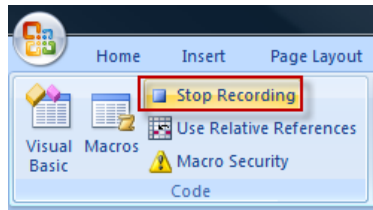


Now that we have the Buttons created, we can build the macros that will actually perform the work. The easiest way to create simple macros is using Excel's "Record Macro" functionality. When we turn on Record Macro, Excel records all our actions and creates a new subroutine (or program snippet). The first one that we want to record is the "View" button for the Profit and Loss Statement. For this small program, we just want Excel to change the active worksheet from the UserControl worksheet to the Profit and Loss Statement worksheet (ProfLoss on the worksheet tab). We also want to think about where the cursor will be positioned when the user is switched to the new worksheet.

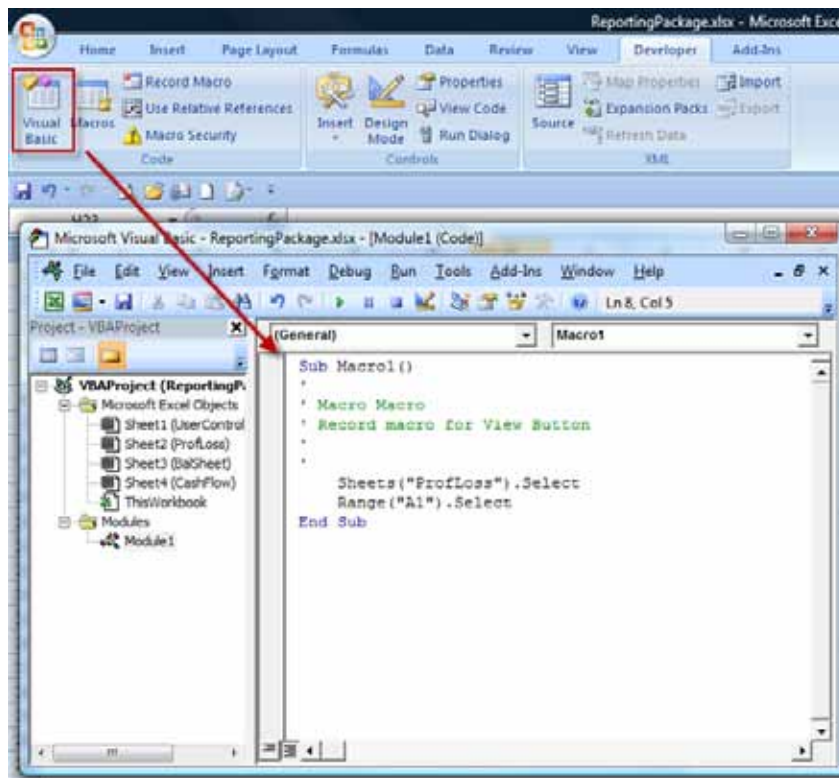
In order to record commands for the "View" Button, we need to be on the UserControl worksheet. Next, we navigate to the Developer tab, and select Record Macro in the Code group. Here we have just added "Record macro for View Button" in the description. Once we have reached this point, we select OK.



Excel is now recording our key strokes and mouse clicks. The first step in the macro is to switch to the ProfLoss worksheet. Just click the worksheet tab at the bottom of the workbook to change to ProfLoss. Once on ProfLoss, press Cntl-Home to home the cursor to the top left on the ProfLoss worksheet. These two commands are all we need to get users from the UserControl worksheet to the Profit and Loss Statement. The next step is to select Stop Recording to stop recording more steps.



Now that we have recorded our macro, we can look at what was actually recorded in Excel's programming object. You need to select the Visual Basic selection on the Code Group, which will appear as follows:



Taking a closer look at our code snippet you can see that there are only two VBA statements in our macro (or Sub for Subroutine), which Excel titled "Macro1()". The VB statements are as follows:

```
(General)

Sub Macro1()
'
' Macro Macro
' Record macro for View Button
'
'
' Sheets("ProfLoss").Select
Range("A1").Select
End Sub
```

Changes to ProfLoss worksheet

Homes cursor at cell A1

Our code is enclosed within the Sub Macro1() and End Sub statements. When we generate the code for the other two View buttons we are going to just copy and paste this subroutine and make a few changes for the other worksheet names. We will name the other two macros Macro2() to view the BalSheet and Macro3() to view the CashFlow worksheet. Once we are done with the changes, our code, with the changes highlighted, appears as follows:

```
(General)

Sub Macro1()
'
' Macro Macro
' Record macro for Profit & Loss View Button
'
'
' Sheets("ProfLoss").Select
Range("A1").Select
End Sub

Sub Macro2()
'
' Macro Macro
' Record macro for View Balance Sheet Button
'
'
' Sheets("BalSheet").Select
Range("A1").Select
End Sub

Sub Macro3()
'
' Macro Macro
' Record macro for View Cash Flow Button
'
'
' Sheets("CashFlow").Select
Range("A1").Select
End Sub
```

Now we just need to assign each of the three macros to the correct View button on the UserControl worksheet. So we navigate back to the UserControl worksheet and start the process. First we right click the View button that is adjacent to the Profit and Loss row on the UserControl worksheet and select the Assign Macro option on the drop down list.




Then assign Macro1 to the first View button and select OK as indicated below.



We follow the same process for the Balance Sheet and Cash Flow View buttons, assigning Macro2 and Macro3 respectively. Then we return to the UserControl worksheet and test the Buttons. When we select the View button for the Profit and Loss statement, we immediately switch to the ProfLoss worksheet. We also note that the cursor is located at cell A1, which was a requirement for our macro. At this point, the View buttons are programmed and tested.

Now we can create and assign the macros for the Print Buttons. The first three Buttons will only print the specific worksheet, e.g. the Balance Sheet. The fourth button will print all three financial statements. We decided that after the Print macro is finished, we wanted to return to the UserControl worksheet and home the cursor at "A1", which is a little different from all the other macros; but more than likely a user that will want to end up at the UserControl worksheet rather than last worksheet to be printed.

To get started, we will use the same process to Record a new macro. Assuming that we have already configured the Print Setup for each of the three worksheets, we can just initiate the recording. Starting from the UserControl worksheet, select the ProfLoss worksheet, then select Print  to print the document and return to UserControl by selecting the UserControl tab at the bottom of the worksheet and press Cntl-Home. Returning to the Visual Basic selection on the Developer menu tab, we can see the recorded macro, which appears as follows:


```

Sub Macro4()
'
' Macro4 Macro
' This macro Prints the Profit and Loss Statement
'
    Sheets("ProfLoss").Select
    ExecuteExcel4Macro "PRINT(1,,1,,,,,,2,,,TRUE,,FALSE)"
    Sheets("UserControl").Select
    Range("A1").Select
End Sub

```

Selects the worksheet for Profit and Loss
 Prints current worksheet
 Returns to UserControl worksheet
 Home cursor

Now we need to duplicate this snippet for Macro5() to print our Balance Sheet and Macro6() to print our Cash Flow Statement. Printing all the worksheets with Macro7() is obviously just a consolidation of Macro4(), Macro5() and Macro6(), and returning to the UserControl worksheet and homing the cursor as the final two steps. It will appear as follows:

```

Sub Macro7()
'
' Macro
' This macro Prints all three statements
'
    Sheets("ProfLoss").Select
    ExecuteExcel4Macro "PRINT(1,,1,,,,,,2,,,TRUE,,FALSE)"
    Sheets("BalSheet").Select
    ExecuteExcel4Macro "PRINT(1,,1,,,,,,2,,,TRUE,,FALSE)"
    Sheets("CashFlow").Select
    ExecuteExcel4Macro "PRINT(1,,1,,,,,,2,,,TRUE,,FALSE)"
    ' Return to UserControl
    Sheets("UserControl").Select
    Range("A1").Select
End Sub

```

That is about it for our simple user control example. The View buttons may seem of little value, but when you have a workbook that contains 27 worksheets, having a button that will immediately navigate the user to the desired worksheet is a huge time saver. The print functions speak for themselves. Hopefully this little macro primer piqued your interest enough to learn more about using Visual Basic in your applications.

Tip #8 -- Dynamic reporting tools

Background

Another handy function to familiarize yourself with is the CHOOSE() function. CHOOSE() is extremely useful given that you can use it to return character strings, values, and even other functions, based on an index. Frequently with clients, we will use the CHOOSE() function to create a financial reporting application that allows users to very easily modify the inclusive dates used to generate reports. It can also help with charts, allowing you to easily control the data being provided to those charts. For our application below, we are only going to illustrate how to use the CHOOSE() function in a financial reporting application for revenue and standard margin analyses.

Application

The format for the CHOOSE function is as follows:

`CHOOSE(index_number, value1, value2,...)`

On the surface, this function does not appear to be particularly helpful. But it can be very helpful when you want to give report users the ability to change reporting periods for financial analysis. For example, you can develop reports that allow users to easily change time periods, and/or comparison baselines (i.e. budget vs. prior year). Using this approach, users can select the time period, e.g. 4 for April, and then the report populates with April Plan and April actual data for 2011 as well as the actual April 2010 prior-year data and then recalculates the variances. In our next application, we have included six supporting worksheets and two analysis worksheets. The analysis worksheets reference at least two of the supporting worksheets.

Using this application, a client can analyze revenue variances for volume and price, as well as bridge a standard margin variance for price, volume, cost and mix. Our hypothetical company has divided revenue reporting into three product families. Within each product family there are two or more products. For example, a product family for an electronics manufacturer might be televisions. Then within a product family, you would have the different television models, e.g. 30", 40", etc. The next product family could be DVD players, with the different models beneath the DVD family (our example just uses three generic Product Families--A, B and Accessories/Other). Our worksheets will have, at the model or product level, total dollars shipped, units shipped and per unit dollars for a particular dimension (the dimensions are revenue and cost of goods). Descriptions of the six worksheets are as follows:

Pln_RV (Revenue Plan Summary): This is the 2011 revenue detail plan. It contains model-level revenue including average selling prices (ASP), units shipped and the resulting revenue dollars, in thousands (\$K). The first model number—Product A.1—is referenced below.

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Product Family A														
Product A.1	Revenue (\$K)	\$4,500.0	\$4,800.0	\$5,600.0	\$5,500.0	\$6,100.0	\$4,800.0	\$7,000.0	\$7,500.0	\$6,500.0	\$7,900.0	\$6,200.0	\$5,800.0	\$72,200.0
	Units	2,727	2,909	3,394	3,333	3,697	2,909	4,242	4,545	3,939	4,788	3,758	3,515	43,758
	ASP	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650	\$1,650

Pln_CG (Planned Cost of Goods): This is the planned cost of goods (COGS) at standard price. Standard price is the product cost established for a particular item. The difference between revenue and standard price represents standard margin or profit. Similar to the planned revenue worksheet, this is the planned cost of goods by model, including the total cost of goods shipped, units shipped (same as revenue forecast) and standard unit cost.

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Product Family A														
Product A.1	Cost of Goods	\$2,045.5	\$2,181.8	\$2,545.5	\$2,500.0	\$2,772.7	\$2,181.8	\$3,181.8	\$3,409.1	\$2,954.5	\$3,590.9	\$2,818.2	\$2,636.4	\$32,818.2
	Units	2,727	2,909	3,394	3,333	3,697	2,909	4,242	4,545	3,939	4,788	3,758	3,515	43,758
	Std Cost	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750

Pln_SM (Planned Standard Margin): This is just the difference between the revenue generated and the standard cost, which is the standard margin or profit for a particular product. On this worksheet we included the gross profit or standard margin % and unit contribution margin (per unit revenue less per unit standard cost).

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Product Family A														
Product A.1	Gross Profit (\$K)	\$2,454.5	\$2,618.2	\$3,054.5	\$3,000.0	\$3,327.3	\$2,618.2	\$3,818.2	\$4,090.9	\$3,545.5	\$4,309.1	\$3,381.8	\$3,163.6	\$39,381.8
	Gross Profit %	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%	120.0%
	Unit Cost Margin	\$900	\$900	\$900	\$900	\$900	\$900	\$900	\$900	\$900	\$900	\$900	\$900	\$900

Act_RV (Actual Revenue): This is the actual revenue, units shipped and ASP. Typically, on the plan side, we forecast using the estimated ASP and units to be shipped to derive the planned revenue. On the actual revenue side, our ERP system provides the revenue dollars billed and the actual units shipped. With that information we derive the actual ASP. We only have actual revenue data through April 2011.

		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Product Family A														
Product A.1	Revenue (\$K)	\$4,600.0	\$5,200.0	\$5,400.0	\$5,100.0									\$20,300.0
	Units	2,500	3,100	3,400	3,100									12,100
	ASP	\$1,840	\$1,677	\$1,588	\$1,645	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$1,678

Act_CG (Actual Cost of Goods): This is the cost of goods associated with shipping a particular model number. Again, this information comes from our ERP system and represents units shipped to customers and the associated cost at standard. Again, we only have actual cost of goods through April 2011.

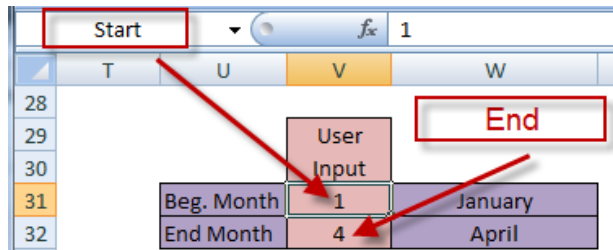
		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
Product Family A														
Product A.1	Cost of Goods	\$1,925.0	\$2,309.5	\$2,550.0	\$2,356.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$9,140.5
	Units	2,500	3,100	3,400	3,100	0	0	0	0	0	0	0	0	12,100
	Std Cost	\$770	\$745	\$750	\$760	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$750	\$755

Act_SM (Actual Standard Margin): This is just the actual revenue, minus the actual cost of goods sold, which represents the standard margin or profit for a particular model number, through April 2011.

			Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Total
4															
5		Product Family A													
7		Product A.1	Standard Margin (\$K)	\$2,675.0	\$2,890.5	\$2,850.0	\$2,744.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$11,159.5
8			Standard Margin %	139.0%	125.2%	111.8%	116.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	122.1%
9			Unit Cont Margin	\$1,070	\$992	\$838	\$885	\$0	\$0	\$0	\$0	\$0	\$0	\$0	\$922

Analysis Worksheets

Most of the work is performed on the two analysis worksheets (Rev_Anlys and SM_Anlys). Users can select the periods to be evaluated. Using the area shown below on the Rev_Anlys worksheet, users can select the analysis period by inputting the numeric start and end months. If both months are the same, then the report will update for that specific month. Note that we are making use of Defined Name references in this example. The Defined Name for V31 can be seen in the Name Box to the left of the formula bar in the snapshot below. For this example, the Defined Name reference in the Name Box, when V31 is selected, is "Start". [See Tip #2 for more information on using Defined Names.] Also, we named cell V32 "End".



In column W above, we begin using our CHOOSE() function to display the month that corresponds to the numeric month in column V. The formulas at W31 and W32 are as follows:

W31 =CHOOSE(Start, "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December")
W32 =CHOOSE(End, "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December")

We are using Named References as an index for our CHOOSE() function to make the formula more readable. We used Excel's Camera function to take a snap shot of the User Input area and pasted our image to the top of the report, as seen below:



The example above shows how to use the CHOOSE() function to return a text string. The CHOOSE() function can also be used to return values, which is what we want to do next.

Rev_Anlys (Revenue Analysis): The revenue analysis worksheet is where the supporting worksheets come together. As you can see below, the report is broken into four horizontal sections, which are Plan, Actual, Delta and Variance Analysis.

In the Plan section, we are going to reference the planned revenue data on Pln_RV worksheet. The Actual section will reference the Act_RV worksheet. The Delta and Variance Analysis section are all calculations based on the Plan and Actual sections.


=SUM(CHOOSE(Start,value1,value2,...value12):CHOOSE(End,value1,value2...value12))

=SUM(CHOOSE(Start,F7,G7,...,Q7);CHOOSE(End, F7,G7,...,Q7))


[illegible]

The Actuals worksheet (Act_RV) only contains results through April 2011. When we change the end month to the value 4, we will get our year-to-date revenue results. The Variance Analysis section performs a simple price vs. volume variance analysis. The price variance is calculated by subtracting the plan ASP from the actual ASP and multiplying by the actual

shipment volume (or (column L – column H) * column K). The resulting year-to-date April 2011 revenue report is as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2				Revenue Summary															
3																			
4																			
5				Product Family A															
6																			
7				Product A.1															
8				Product A.2															
9				Total Family A															
10																			
11				Product Family B															
12																			
13				Product B.1															
14				Product B.2															
15				Total Family B															
16																			
17				Accessories & Other Adj															
18																			
19				Accessory #1															
20				Accessory #2															
21				Other Adj.															
22																			
23				TOTAL Revenue															

If we want to convert this report to an April report, just change the Start month to the value 4 and voila, we have a revenue report for the month of April 2011.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1																			
2				Revenue Summary															
3																			
4																			
5				Product Family A															
6																			
7				Product A.1															
8				Product A.2															
9				Total Family A															
10																			
11				Product Family B															
12																			
13				Product B.1															
14				Product B.2															
15				Total Family B															
16																			
17				Accessories & Other Adj															
18																			
19				Accessory #1															
20				Accessory #2															
21				Other Adj.															
22																			
23				TOTAL Revenue															

SM_Anlys (Standard Margin Analysis): The standard margin worksheet contains three sections, which include Gross Profit, Price Variances and Volume & Mix Variances. The variance column adjacent to the Gross Profit Plan and Actual columns is the total gross profit variance.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2	<i>Standard Margin Analysis</i>														
3				Gross Profit			Price Variances		Volume & Mix Variances						
4				Plan	Actual	Variance	Sales	Cost	Sales	Cost	Final	Mix			

By using the CHOOSE() function to summarize revenue, cost of sales, standard margins and units, we are easily able to parse our standard profit (revenue less standard cost) into different variance categories to provide additional insight into what is driving profit variances. The variance categories are as follows:

Price Variance – Sales	Variance generated by differences between budgeted and actual sales prices, times the actual sales
Price Variance – Cost	Variance generated by differences between budgeted and actual costs, times the actual sales
Volume Variance – Sales	Variance generated by volume differences between the budgeted and actual volumes, times the budgeted sales prices.
Volume Variance – Cost	Variance generated by volume differences between the budgeted and actual volumes, times the budgeted costs.
Volume Variance – Final	When analyzing family-level or total Cost and Sales volume variances, we need to parse the combined Cost and Sales volume variance into Mix and Final volume variances; otherwise, you will lose sight of the actual volume impact at a summary level. So this is the net remainder once the mix component is removed from the total sales and cost volume variance.
Mix Variance	Mix represents the impact of volume changes between products or items with different per unit contribution margins within a group. When we summarize the results of two or more products, we can have a situation where units are favorable (or volume), but results are unfavorable on margin dollars. This is because the item that is producing the favorable volume has a lower per unit contribution margin, offsetting the entire favorable volume impact. The mix variance calculation separates the mix from the volume, which in our example would produce a favorable volume variance and unfavorable mix.

At a total standard margin level analysis, the Cost and Sales volume variances are subsumed by the Mix and Final volume variances. The total standard margin analysis, for the month of April 2011, is as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2	Standard Margin Analysis														
3				Gross Profit			Price Variances		Volume & Mix Variances						
4				Plan	Actual	Variance	Sales	Cost	Sales	Cost	Final	Mix			
5	Product Family A														
7	Product A.1			\$3,000.0	\$2,744.0	(\$256.0)	(\$15.0)	(\$31.0)	(\$385.0)	\$175.0					
8	Product A.2			\$1,034.5	\$1,310.0	\$275.5	\$100.0	\$10.0	\$400.0	(\$234.5)					
9	Total Family A			\$4,034.5	\$4,054.0	\$19.5	\$85.0	(\$21.0)			\$33.9	(\$78.4)			
11	Product Family B														
13	Product B.1			\$612.5	\$925.0	\$312.5	\$50.0	\$17.5	\$700.0	(\$455.0)					
14	Product B.2			\$93.8	\$0.0	(\$93.8)	\$0.0	\$0.0	(\$500.0)	\$406.3					
15	Total Family B			\$706.3	\$925.0	\$218.8	\$50.0	\$17.5			\$84.8	\$66.5			
17	Accessories & Other Adj														
19	Accessory #1			\$5,000.0	\$4,816.0	(\$184.0)	\$135.0	\$31.0	(\$385.0)	\$35.0					
20	Accessory #2			\$20.6	\$9.1	(\$11.6)	(\$9.5)	(\$0.7)	(\$1.8)	\$0.4					
21	Other Adj.			\$15.0	\$0.0	(\$15.0)	\$0.0	\$0.0	(\$15.0)	\$0.0					
22	Total Accessories & Other Adj			\$5,035.6	\$4,825.1	(\$210.6)	\$125.5	\$30.3			(\$353.5)	(\$12.9)			
24	TOTAL			\$9,776.4	\$9,804.1	\$27.7	\$260.5	\$26.8			(\$234.8)	(\$24.8)			

Again, now that we have set up the CHOOSE() function to select our values, all we need to do is change the inclusive dates for our report, perhaps changing it to start at 1 for January and end at 3 for March, which generates a 1Q 2011 margin report.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2	Standard Margin Analysis														
3				Gross Profit			Price Variances		Volume & Mix Variances						
4				Plan	Actual	Variance	Sales	Cost	Sales	Cost	Final	Mix			
5	Product Family A														
7	Product A.1			\$8,127.3	\$8,415.5	\$288.2	\$350.0	(\$34.5)	(\$50.0)	\$22.7					
8	Product A.2			\$4,058.0	\$3,557.5	(\$500.5)	\$42.1	(\$97.5)	(\$1,042.1)	\$597.0					
9	Total Family A			\$12,185.3	\$11,973.0	(\$212.3)	\$392.1	(\$132.0)			(\$578.4)	\$106.0			
11	Product Family B														
13	Product B.1			\$2,655.0	\$3,622.5	\$967.5	\$575.0	\$186.0	\$525.0	(\$318.5)					
14	Product B.2			\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0	\$0.0					
15	Total Family B			\$2,655.0	\$3,622.5	\$967.5	\$575.0	\$186.0			\$206.5	\$0.0			
17	Accessories & Other Adj														
19	Accessory #1			\$13,545.5	\$11,930.0	(\$1,615.5)	(\$150.0)	\$80.0	(\$1,700.0)	\$154.5					
20	Accessory #2			\$56.4	\$82.2	\$25.8	\$26.5	(\$2.1)	\$1.8	(\$0.4)					
21	Other Adj.			(\$20.0)	\$0.0	\$20.0	\$0.0	\$0.0	\$20.0	\$0.0					
22	Total Accessories & Other Adj			\$13,581.8	\$12,012.2	(\$1,569.7)	(\$123.5)	\$77.9			(\$1,511.8)	(\$12.3)			
24	TOTAL			\$28,422.1	\$27,607.7	(\$814.5)	\$843.6	\$131.9			(\$1,883.7)	\$93.7			

As you can see from our revenue and standard margin analysis reports, using the CHOOSE() functions will provide your users with a tremendous amount of control over their analysis. Enabling your users to more effectively analyze variances and determine if corrective action is required will add significant value to their customers.

Tip #9 -- Using customized lists to increase productivity

Background

When working one on one with our clients, we frequently show them how to create their own customized lists. This is a simple task to perform, but it can save users a great deal of time and effort. The most common application of this tip applies to users that create many spreadsheets and consequently have to repeat the same column headers over and over. For example, a client's standard practice might be to list the months of the year with breaks for quarter totals and a year total. It might be "Jan 2010" or "Jan '10", and then the quarters are 1Q 2010 or 1Q '10, and so on and so forth.

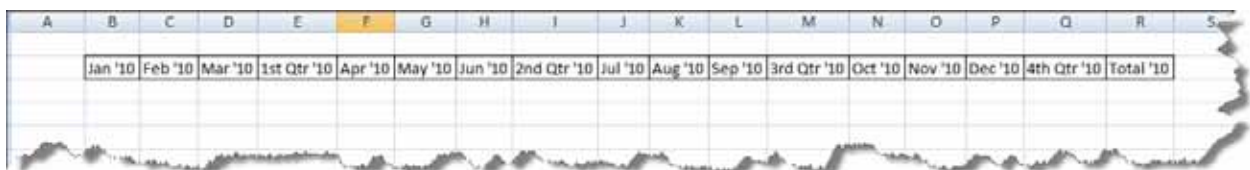
One option to avoid having to manually type in these values for each worksheet is to open the last spread sheet where we created the column headers and perform a copy and paste. Another might be to create a new default Excel template, which opens every time we start Excel, with the correct column headers. Both are good options, but neither is efficient as using customized lists.

Application

The first step is to either open an existing worksheet with our column headers or recreate them on a new worksheet. Our example company uses the following time periods to report 2010 financial results:

Jan '10 | Feb '10 | Mar '10 | 1st Qtr '10 | Apr '10 | to |Dec '10 | 4th Qtr '10 | Total '10 |

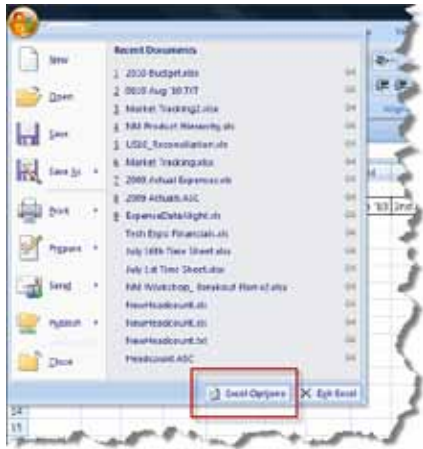
Our custom list displayed on a worksheet appears as follows:



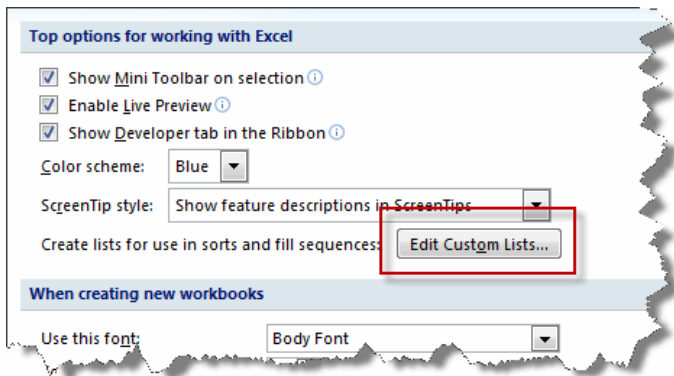
The screenshot shows an Excel worksheet with a custom list of time periods for 2010. The list is displayed in a single row across columns A through R. The list items are: Jan '10, Feb '10, Mar '10, 1st Qtr '10, Apr '10, May '10, Jun '10, 2nd Qtr '10, Jul '10, Aug '10, Sep '10, 3rd Qtr '10, Oct '10, Nov '10, Dec '10, 4th Qtr '10, and Total '10. The list is highlighted with a light blue background.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Jan '10	Feb '10	Mar '10	1st Qtr '10	Apr '10	May '10	Jun '10	2nd Qtr '10	Jul '10	Aug '10	Sep '10	3rd Qtr '10	Oct '10	Nov '10	Dec '10	4th Qtr '10	Total '10		

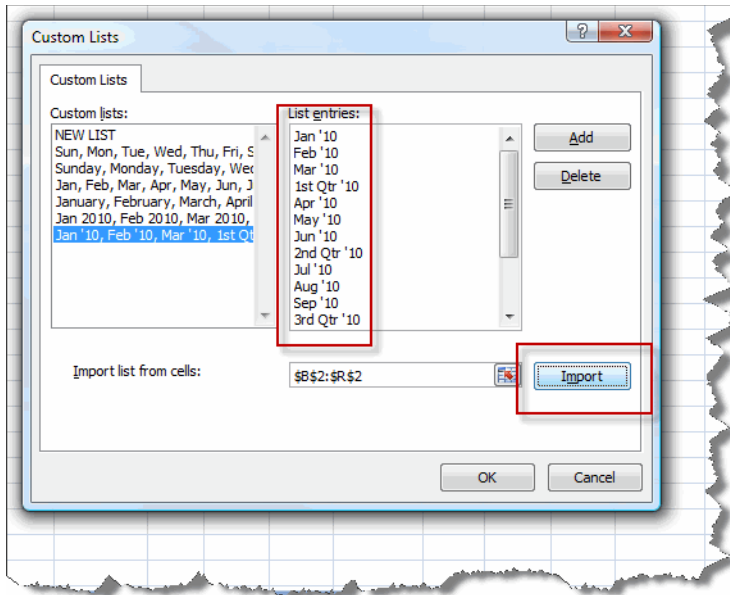
Once we have our custom list displayed, we just need to add the list to Excel's "Custom List" database. First highlight the entire list and then select the "Excel Options" from the ribbon as follows:



Then we select the "Edit Custom Lists" option below:



This will bring up the Custom Lists dialog box. The range that we selected is now referenced in the "Import list from cells" box. Assuming that we have the correct range referenced in the "Import list from cells" box, we then select "Import" to add our custom list to Excel's Custom List database.



We now select OK to save our custom list. To use our custom list, we open a new workbook and type in "Jan '10" which is the first member of our new custom list. We can then grab the lower right-hand corner of the cell and drag our mouse across the area to be filled.



Once we drag the first entry from our Custom List, Excel then automatically fills in the other members for us as displayed below.

	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1																	
2																	
3																	
4																	
5	Jan '10	Feb '10	Mar '10	1st Qtr '10	Apr '10	May '10	Jun '10	2nd Qtr '10	Jul '10	Aug '10	Sep '10	3rd Qtr '10	Oct '10	Nov '10	Dec '10	4th Qtr '10	Total '10

The other nice thing about using Custom Lists is that you can start anywhere in your list. If you just wanted the second half of the year, you could just start with "Jul '10." You do not have to begin with "Jan '10."

Tip #10 – Creating User Defined function macros

Background

User Defined function macros are generally small Visual Basic programs, typically created by advanced Excel users, to perform complex tasks. User Defined function macros are provided parameters required to perform a defined task and return the result to that cell. In theory, a User Defined function macro could be written that simply adds two numbers. If the name of our function macro was AddValues and it had two parameters, it would look like the following:

```
=AddValues(1,2)
```

The result of the above, assuming that it actually adds the two values, would be the value 3, returned to the reference cell. When set up correctly, function macros can be made accessible through Excel's "Insert Formula" dialog when a user selects the "User Defined" function list. Obviously, there is little need for a function macro that adds two values together. But there are many instances where an algorithm or process is just too complex to handle with Excel's built-in functionality. That is where User Defined function macros really start to add value.

To create User Defined function macros, you will need to possess some basic VB programming skills. There are a number of good books on programming Excel applications with VB. Some code can be created using Excel's Macro Recording functionality. But to really unleash the power of function macros, you will need to generate some computer code.

For this primer, we are going to create a fairly simple function macro. Our function macro example will take phrases and convert them into Pig Latin. Though our example is a bit nonsensical, it should begin to give you a feel for what is possible with function macros.

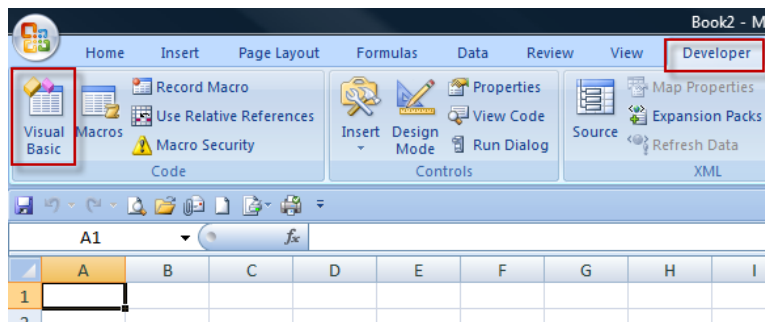
We are using a simple version of Pig Latin, which entails taking words that begin with a consonant, moving that consonant to the end of the word and adding "ay" to the end. So the word "down" becomes "own-day" in Pig Latin. Words beginning with a vowel, get the same treatment with the first letter, but "way" is added to the end. For example, the phrase "come on down" is translated into "ome-cay n-oway own-day". So think about how you would process a text string in Excel, without using VB, to achieve the same result. It's going to be a little complicated.

A few words about User Defined function macros and security. As we know, many virus attacks have been launched using macro code in various file types, especially those that pertain to Outlook. Just the same, someone could, and we are sure many already have, written viruses as part of an Excel workbook that would launch Outlook and start broadcasting itself to everyone in that user's contact list, clogging up the network; consequently, Microsoft has implemented quite a bit of security around VB macros. When distributing User Defined function macros, macro authors need to be familiar with digital

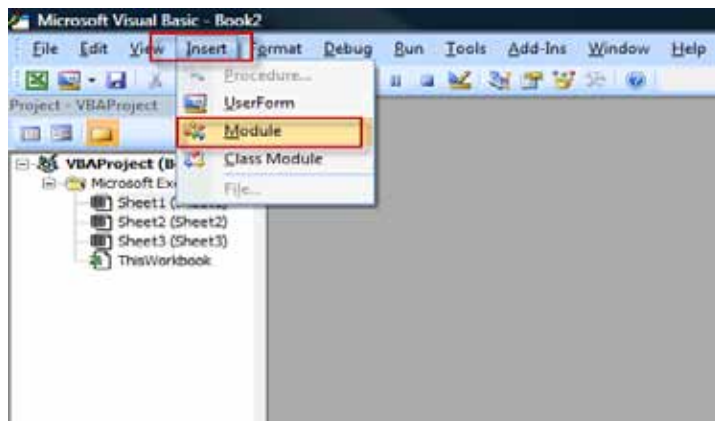
signatures and how users can trust a digital signature. Keep in mind that there will be more effort required to actually push out a function macros to users when the time comes.

Application

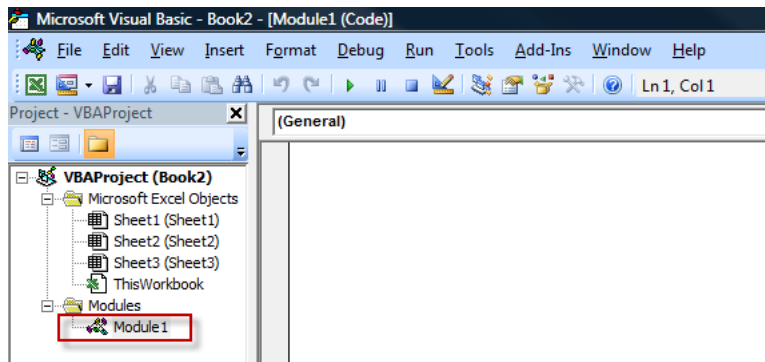
Similar to “Tip #6 – Setting up simple user controls” we are going to have to dive into Excel’s Visual Basic editor again. Choose the Developer menu tab and then select the Visual Basic from the Code group as follows:



You will then need to Insert a module, which is similar to adding a text document, that will be used to generate our VB code.



Once you add the Module, your screen should appear as follows, with Module1 referenced in the left-side margin:



As mentioned earlier, when we create a function macro we are adding to Excel's function library. We decided that the name of our new User Defined function macro is going to be PigLatin(). Once we have the function macro created, then we can add it to a cell, e.g. =PigLatin(A10) at cell B10, and assuming that there is a phrase at A10, our phrase will be translated into pig latin at cell B10.

As you may recall from Tip #6, we created subroutines that started with Sub Marco1() and ended with End Sub. Our User Defined function macro begin as follows:

Function PigLatin(OriginalString) As String

The word "Function" tells Excel that this is a function macro, rather than a subroutine (Sub) as was used in Tip #6. Then we tell Excel that the function name is PigLatin, that we will provide our function with one argument—OriginalString—and that our function will return a text string. We also provide some simple remarks about what our program is going to accomplish and we provide a Function End statement to identify the end of our function macro, which appears as follows:

```
Function PigLatin(OriginalString) As String
'This is a Pig Latin converter. If the first letter of a word is a consonant,
'move it to the end and add "ay". If the first letter of a word is a vowel,
'move it to the end and add "way"
```

```
End Function
```

Our first phrase is going to be "come on down", which will be at cell A10. The function macro will be inserted at cell B10 (=PigLatin(A10)). When the function macro is evaluated or processed, the string variable OriginalString will be set equal to our "come on down" phrase (OriginalString = "come on down"). Since our algorithm is going to have to evaluate our string one character at a time, we need to know the length of OriginalString, which will be done using the LEN() function. Also, we need to initialize a counter—I1—to help track how many characters have been processed, and a flag—IsFirstLetter—to help identify when we are starting a new word. Now our code snippet appears as follows:

```

Function PigLatin(OriginalString) As String
    'This is a Pig Latin converter.  If the first letter of a word is a consonant,
    'move it to the end and add "ay".  If the first letter of a word is a vowel,
    'move it to the end and add "way"
    StringSize = Len(OriginalString)
    I1 = 1
    IsFirstLetter = "Yes"

End Function

```

Our function macro will use a Do While statement to work through every character until we reach the end of our phrase or text string. To accomplish this, the Do While is set to loop until we have exceeded the length our string by one. On the first pass through the Do While loop, we know that I1 will equal one and IsFirstLetter will equal "Yes". Adding the Do While and first IF statement to check for the first letter to our code snippet appear as follows:

```

Function PigLatin(OriginalString) As String
    'This is a Pig Latin converter.  If the first letter of a word is a consonant,
    'move it to the end and add "ay".  If the first letter of a word is a vowel,
    'move it to the end and add "way"
    StringSize = Len(OriginalString)
    I1 = 1
    IsFirstLetter = "Yes"
    Do While I1 <> StringSize + 1
        NewLtr = Mid(OriginalString, I1, 1)
        If IsFirstLetter = "Yes" Then
            '
        End If
        I1 = I1 + 1
    Loop
End Function

```

Set NewLtr to the next character in OriginalString using MID()

Test to see if this is the first letter of a word. We will add an ELSE clause to process the other characters

Increment counter to to move to next character in OriginalString

Now we need to figure out if we have a consonant or vowel as the first letter of our new word. Once we have determined that, then we can add either "ay" to the end of our vowel or "way" to the end of the consonant. We are going to assign the first letter and suffix to the variable named EndPart. Once we have processed through the word, then we will append EndPart to the other characters and move onto the next word. The section that determines if we have a vowel or not and creates the EndPart is as follows:


```

Function PigLatin(OriginalString) As String
    'This is a Pig Latin converter. If the first letter of a word is a consonant,
    'move it to the end and add "ay". If the first letter of a word is a vowel,
    'move it to the end and add "way"
    StringSize = Len(OriginalString)
    I1 = 1
    IsFirstLetter = "Yes"
    Do While I1 <> StringSize + 1
        NewLtr = Mid(OriginalString, I1, 1)
        If IsFirstLetter = "Yes" Then
            If NewLtr = "A" Or NewLtr = "E" Or NewLtr = "I" Or NewLtr = "O" _
            Or NewLtr = "U" Or NewLtr = "a" Or NewLtr = "e" Or NewLtr = "i" _
            Or NewLtr = "o" Or NewLtr = "u" Then
                'The first letter of the new word is a vowel. Add remove the letter and add "way"
                EndPart = NewLtr + "way"
            Else
                'The first letter of the new word is a consonant. Move to EndPart and add "ay"
                EndPart = NewLtr + "ay"
            End If
            IsFirstLetter = "No"
        Else
            'Reset IsFirstLetter to "No" now that we have
            'processed the first letter of a new word. We will not
            'have to check for a vowel again until the next new
            'word is processed.
        End If
        I1 = I1 + 1
    Loop
End Function

```

If first letter is a vowel, then process this section

If the first letter is not a vowel--it is a consonant-- then process this section

Reset IsFirstLetter to "No" now that we have processed the first letter of a new word. We will not have to check for a vowel again until the next new word is processed.

Next, we just have to process the remainder of our word. The translated phrase will reside in the variable we defined as the function macro name, which is PigLatin. At this point we have determined if our word started with a vowel or consonant, added the appropriate suffix and temporarily stored the result in EndPart. The second letter will be appended to PigLatin until we reach a Space or blank character, then we will append EndPart to the end of our word to get the final pig latin translation. Here is what the next section of code does:

```

Function PigLatin(OriginalString) As String
    'This is a Pig Latin converter. If the first letter of a word is a consonant,
    'move it to the end and add "ay". If the first letter of a word is a vowel,
    'move it to the end and add "way"
    StringSize = Len(OriginalString)
    I1 = 1
    IsFirstLetter = "Yes"
    Do While I1 <> StringSize + 1
        NewLtr = Mid(OriginalString, I1, 1)
        If IsFirstLetter = "Yes" Then
            If NewLtr = "A" Or NewLtr = "E" Or NewLtr = "I" Or NewLtr = "O" _
            Or NewLtr = "U" Or NewLtr = "a" Or NewLtr = "e" Or NewLtr = "i" _
            Or NewLtr = "o" Or NewLtr = "u" Then
                'The first letter of the new word is a vowel. Add remove the letter and add "way"
                EndPart = NewLtr + "way"
            Else
                'The first letter of the new word is a consonant. Move to EndPart and add "ay"
                EndPart = NewLtr + "ay"
            End If
            IsFirstLetter = "No"
        Else
            'This is not the first letter of the word. It may be a letter or space
            If Mid(OriginalString, I1, 1) <> " " Then
                PigLatin = PigLatin + Mid(OriginalString, I1, 1)
            Else
                'If the next space is a blank, then append the EndPart suffix to the string
                PigLatin = PigLatin + "-" + EndPart + " "
                IsFirstLetter = "Yes"
            End If
        End If
        I1 = I1 + 1
    Loop
    PigLatin = PigLatin + "-" + EndPart
End Function

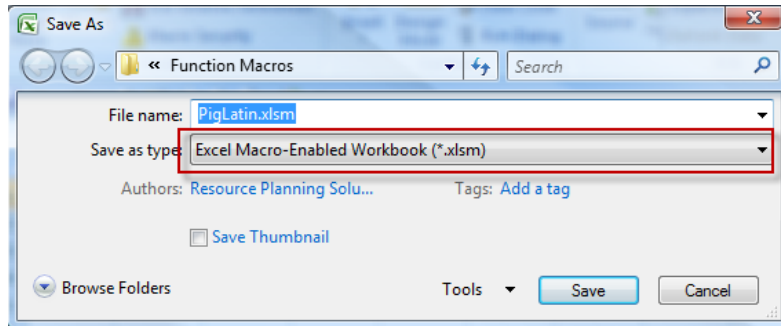
```

If the current character being processed is not equal to a space (" "), then append it onto the end of PigLatin.

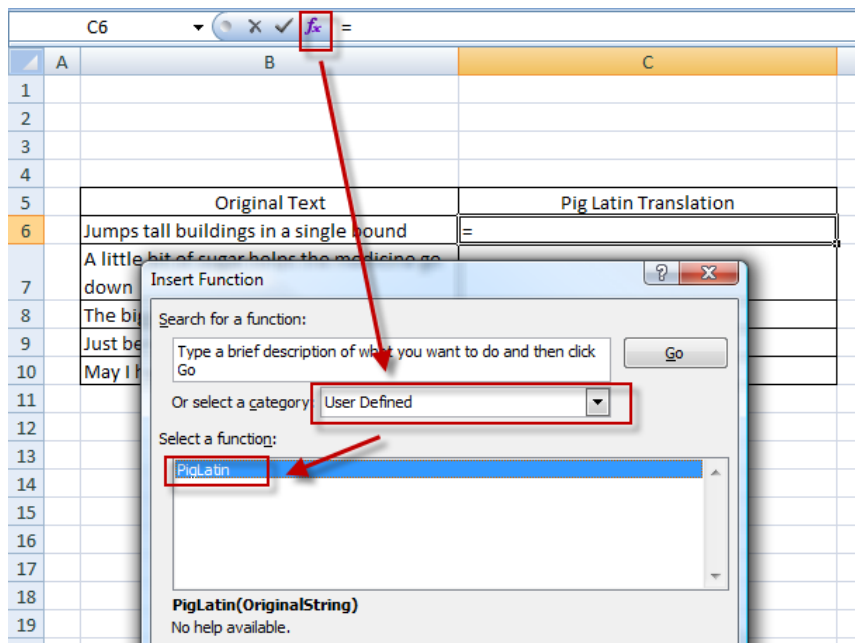
If the current character being processed is a space (or " "), then we want to append the EndPart onto PigLatin, preceded by a hyphen (or "-") and followed by a space (or " ") to separate it from the next translated word.

This last statement only affects the last word being translated. This is required to append the final EndPart to the last word in the phrase.

That is our basic programming example for Function Macros. With the change to Excel 2007, there is a new file type—Excel Macro-Enabled Workbook--that you have to use when saving workbooks that include macros, which appears as follows:



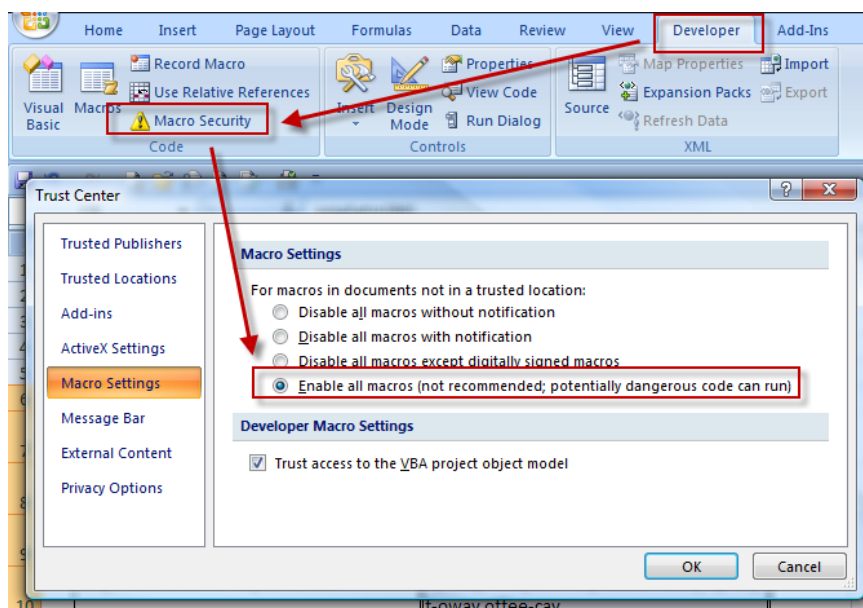
To see how the macro works you need to navigate back to the workbook that is associated with the function macro. Here we have typed in a number of phrases in column B and now need to paste in our PigLatin function macro into column C. Using Excel's insert function dialog, select the User Defined function macro list, of which there is only one—PigLatin(), and paste it into the adjacent cell.



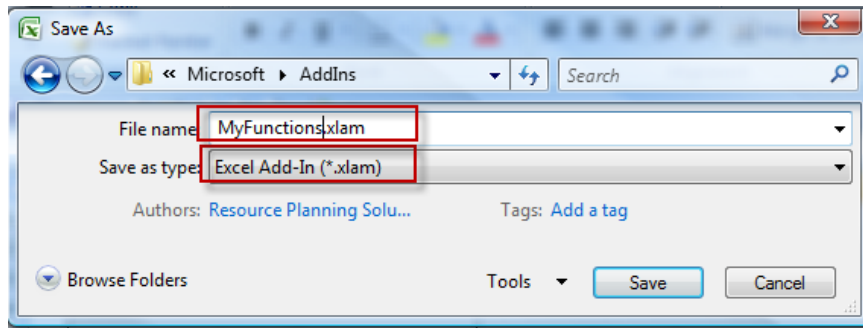
After you hit OK, then just add cell B3 to the reference and the function macro will translate the phrase at B3. Here we have pasted the PigLatin() function down through our list of phrases. It is not perfect, but hopefully you get the idea. There are a number of cases that our simple function macro cannot handle, for example it will have a problem with commas, periods, colons, etc. because we have not programmed logic to catch those issues and take action accordingly.

4		
5	Original Text	Pig Latin Translation
6	Jumps tall buildings in a single bound	jumps-Jay all-tay uildings-bay n-iway -away
7	A little bit of sugar helps the medicine go down	-Away ittle-lay it-bay f-oway ugar-say elps-hay he-tay edicine-may o-gay own-day
8	The bigger they are the harder they fall	he-Tay igger-bay hey-tay re-away he-tay arder-hay hey-tay all-fay
9	Just between you and me this is silly	ust-Jay etween-bay ou-yay nd-away e-may his-tay s-iway illy-say
10	May I have another cup of coffee	ay-May -lway ave-hay nother-away up-cay f-oway offee-cay

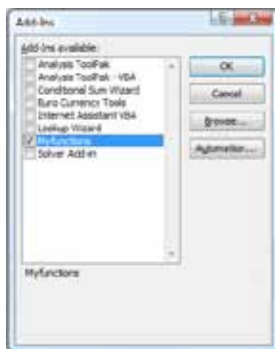
In order to use your function macro, you will have to set the security settings on your computer to enable function macros. Again, you will want to use caution here because you do not want to leave yourself vulnerable to macro viruses. In order to enable the function macro, you will need to navigate to the Developer menu pad, Code group and select "Enable all macros (not recommend; potentially dangerous code can run)", as shown below. By self signing User Defined function macros will permit use of the "Disable all macros except digitally signed macros" option, which is safer.



At this point, the PigLatin() function macro is only available while we have the current workbook open. If you have a User Defined function macro that you always want to be available for your use, then you will need to save the file as an AddIn function. To do this, perform a Save As again and select Excel Add-In option and just use the default directory. Use a generic file name, because you can add more function macros to this work sheet when needed.



Once you have saved the file, select the Microsoft Office Button and choose Excel Options and then AddIns. On the AddIns dialog, select Manage. On the AddIns dialog, select to add in MyFunctions.



Once you have selected OK, you should now be able to access your function macros without having the actual workbook used to create the function macro open. Your function macros will appear now as another section in Excel's function library under User Defined functions.

If this has increased your interest in learning more about VB function macros or just VB in general, check out John Walkenbach's "Excel Power Programming with VBA" book to learn more about using VB with Excel. It is an excellent place to start.

Summary

Well, that is it for our *10 Must Have Tip List*. We hope that you found it enlightening and worthwhile. If you or your team needs more information on how to put these ideas into action, please feel free to contact Resource Planning Solutions at our email address below or call us at 818.436.0781. We may create a few webinars on these skills that would allow us to get into more detail on some of the more complex tips. If you are interested in attending a webinar, send us an email or check our website for updates. Also, if you would like more information about how Resource Planning Solutions can help you turn your vision for financial performance management into a reality, please give us a call.

Cheers from Resource Planning Solutions!

