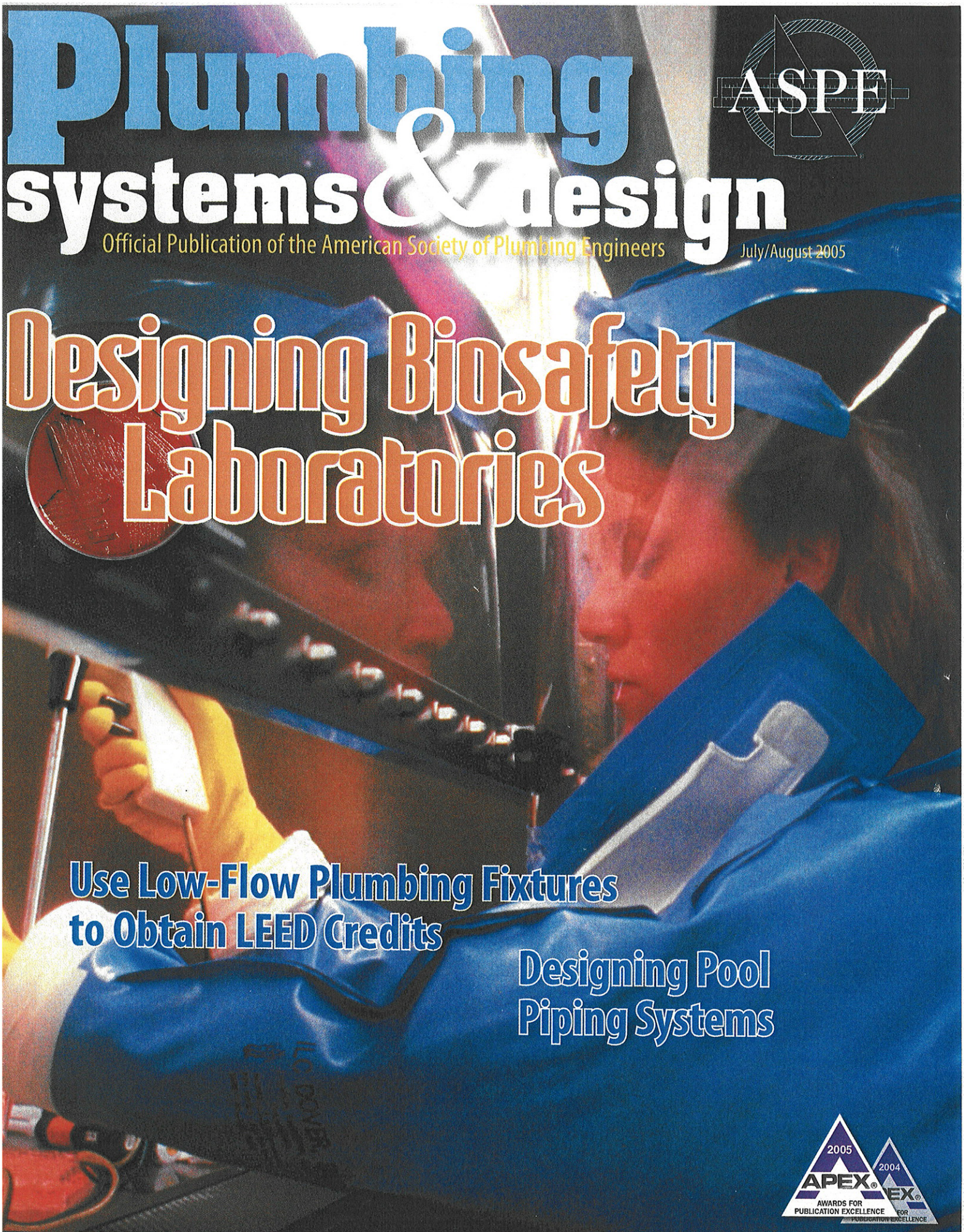
Plumbing & systems & design

Official Publication of the American Society of Plumbing Engineers

July/August 2005



Designing Biosafety Laboratories

Use Low-Flow Plumbing Fixtures to Obtain LEED Credits

Designing Pool Piping Systems

2005 APEX AWARDS FOR PUBLICATION EXCELLENCE

2004 APEX FOR PUBLICATION EXCELLENCE

The magazine for plumbing engineers, designers, specifiers, code officials, contractors, manufacturers, master plumbers, and other plumbing professionals

From the Publisher

Plumbing Systems & Design is the winner and recipient of two 2005 APEX awards. The magazine took top honors in two categories: Technical Writing and Magazines & Journals–Printed Four Color. For the first award, the honor goes to Winston Huff, CPD, LEED AP, for his article "Sustainable-Plumbing-System Technologies for Space and Earth." For the second award, the honor is given to the entire *PSD* staff.

PSD has been publishing for only a short time, since September/October 2002. The magazine is the only one in the plumbing engineering and design industry that can claim the honor of having won an APEX award for every year it has been published! This is a great achievement for the wonderful and hard-working staff, authors, and editors (oh yes, and the publisher too).

This magazine also can lay claim to being well on its way to accomplishing the very purpose for its creation: providing best practices and current and timely technical material directed to plumbing engineers, designers, and contractors. This issue's cover feature, "Designing Biosafety Laboratories," is a good example. For the past century or more, the world has been blessed, or maybe just been lucky, that a major catastrophe in the form of an unstoppable bug getting loose in the world has not occurred, despite all of the research conducted by biological and pharmaceutical companies. With ongoing contracts from the federal government for defensive and offensive weaponry, and public and private research delving into the DNA structure of every living entity on our planet, it's a wonder that it remains safe to breathe the air and drink the water. The thanks go to our plumbing engineers and designers who are responsible for creating safe environments for inhospitable bugs and chemicals.

For all our readers, there's also "A Primer on Liability for Plumbing Professionals" by Steven Nudelman. No one wants to be sued, and Steve, in this second part of his three-part series, helps us understand what a professional must watch out for to avoid negligence, malpractice, fraud, misrepresentation, deceptive practices, and tortuous interference. Steve doesn't provide direct answers. (As he says, "Attorneys give legal advice; magazine articles do not."). What he does provide is an overview of claims issues that plumbing engineering and design professional face on a daily basis. Ignore at your peril.

Anjian Lu, CPD, in his "Computer Corner" column, continues exploring special Excel features that are handy in the engineering profession. In this issue, Lu shows how to create custom menus to go along with custom applications or macros as add-ins to the Excel interface. He actually makes it pretty simple to tackle some fairly complex and advanced issues in using and making the most of the software.

If you still have a mind for the math, check out David Hague's "Focus on Fire Protection" column. In Part 3 of his series on hydraulic design requirements for storage applications, David goes into area and density requirements based on commodities classifications and storage configurations. As he makes clear, "Hydraulic requirements for storage occupancies are not based on the occupancy classification; they are based on the type of commodity protected and the configuration of the storage arrangement." A little light reading for a midsummer night? Not!

For those who just have to work with formulas, check out the Continuing Education section on the fundamentals of domestic water heating. Read the article and take the small exam to earn a continuing education credit. It couldn't be easier, right? These continuing education articles are designed to take about an hour to read and complete the exam and are a unique benefit of *PSD*.

When you're ready for a little lighter reading, check out "Lessons Learned" by Joe Scott. His discovery this issue is that "the time of the year that you preheat the cold water to the heater is the same time of the year that the cold water temperature generally rises anyway: the summer." His lesson? It's "having knowledge of some of a system's peculiar aspects can save [building owners'] money."

If you're ready for a little international journey, check out the "International Digest." From Brazil comes "Application of a Vacuum Toilet System in a Rio de Janeiro Airport." Authors O. Oliveira Jr. and J. Silva Neto summarize the benefits of vacuum toilet systems as significant water usage savers when compared to the more typical gravitational toilets. As they conclude, "It is difficult to imagine any other action that could lead to better results."

While on the issue of saving, Winston Huff in his "Using Low-Flow Plumbing Fixtures to Obtain LEED Credits" article for "Plumbing Technology of the Future" examines the latest generation of low-flow fixtures. As he notes, the current crop of fixtures in no longer in response to a federal mandate but more a response to owners' and designers' interest in voluntary energy-saving programs such as LEED. Winston takes a look at dual-flush water closets, low-flow water closets, low-flow showerheads, and even some water-free products. His mission is clear: "The industry is growing and changing every day, and plumbing engineers need to keep ahead of the curve."

Every issue of *PSD* is jam-packed with technical articles of educational and professional interest. There's the "World of Design/Build," "Peer to Peer," "Plumbing Design by the Numbers," and "Designer's Notebook." "Hydronics for Plumbing Engineers" and "Code Update" help round out the technical material in this issue. Subjects covered include "Designing Zone Pumping Systems" to "A Foot Is not Always a Foot: Disparities in Code Terminology," to "Pool Piping Primer" and "What Plumbing Engineers Should Know About Radon Gas."

It is a joy and a pleasure for all of us to continue to bring you this award-winning magazine. ◆

Stanley Watson

Computer-Aided Hot-Water Circulation System Sizing

Anjian Lu, CPD

Author's Note: *The following column describes an Excel program I wrote to make hot-water circulation system sizing easier. For a free copy of the template contact me at luaj@yahoo.com. Upcoming "Computer Corner" columns will discuss the techniques used in developing this template and other programs relating to plumbing system design.*

In his article "Hot-Water Recirculation Systems" in *Plumbing System & Design*'s May/June 2004 issue, Joseph Messina summarized the importance of hot-water circulation system sizing with practical, simplified methods. Sometimes, though, designers oversize hot-water recirculation systems in an effort to save energy. Thus, I want to introduce a computer-aided sizing procedure that can make sizing more accurate.

How It Works

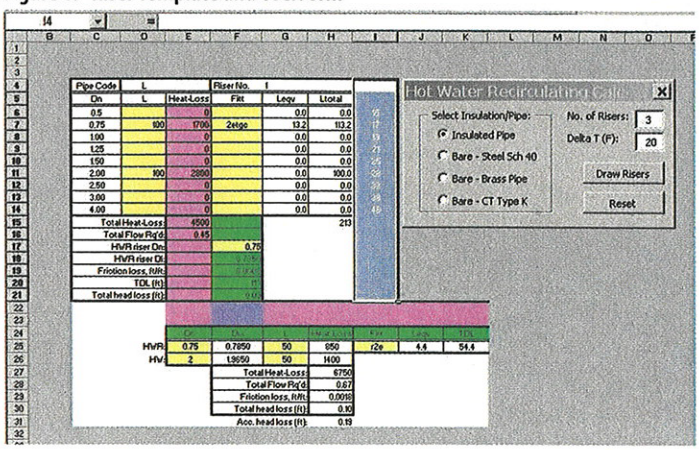
Microsoft Excel is one of the most popular spreadsheets for engineering calculations. Its User-Defined Functions (UDFs) and Visual Basic for Applications (VBA) offer numerous capabilities that simply calculate engineering problems.

Excel has many functions ready for use under the Paste Function icon on the standard toolbar. However, they are not advanced enough for engineering use, such as when you want to calculate friction loss using the Hazen-Williams Formula. A UDF is a function that you can define to perform a job like this. VBA is Microsoft's common programming (macro) language. By using UDFs and VBA you can design a single spreadsheet to calculate hot-water recirculation system sizing.

The Procedures

To perform the calculation, you need to prepare an Excel template with UDFs and VBA procedures and save it in the template directory. In this case the template is called HWRcalc.xlt. Each time you use the program, Excel makes a copy from the template, so you can use it repeatedly without overwriting the original file.

Figure 1. Riser Template and UserForm



When you open the file, a UserForm (a VBA component) appears with a hot-water recirculation system sizing calculation template (see **Figure 1**).

The UserForm. On the right side of the screen is the UserForm, where you can select the pipe type: insulated or bare. If it is bare, select one of the three choices: Bare – Steel Sch 40, Bare – Brass Pipe, or Bare – CT Type K. You also can specify the number of risers (the default is one) and delta T (the default is 20). The reset button in the UserForm restarts new calculations. The draw riser button calculates a simulated riser diagram (explained later).

The Template. On the left side of the screen is the template simulating a typical hot-water recirculation system riser diagram. The pink cells indicate the hot water supply, and the green cells indicate the return. All yellow cells are for user input.

Cell D4 shows the pipe code, a letter indicating the pipe's material and schedule. If you select insulated pipe in the UserForm, the pipe code is L, which means copper tube type L. The code for Bare – Steel Sch 40 is S; the code for Bare – Brass Pipe is B; and the code for Bare – CT Type K is K.

Range C5:C14 lists different pipe sizes. In range D5:D14 you input the riser length of different diameters. Range E5:E14 shows the heat loss, calculated by multiplying the physical riser length in column D by the heat loss per unit pipe length. To calculate the head loss, the program uses the total length (physical or developed length) plus the equivalent length of valves and fittings. When you input the fittings in range F5:F14, their equivalent lengths appear in range G5:G14. In this example there are two elbows, one tee, one gate or ball valve, and one check valve in the hot water return riser. Therefore, you enter 2etgc in cell F7. Range H5:H14 shows total pipe lengths. You can see range I6:I14 if it is highlighted. These figures are specific heat losses for different pipe sizes in Btu/hr per lineal foot when water temperature is 140°F and room temperature is 70°F (see **Table 1**). Range F4:G4 automatically numbers the risers.

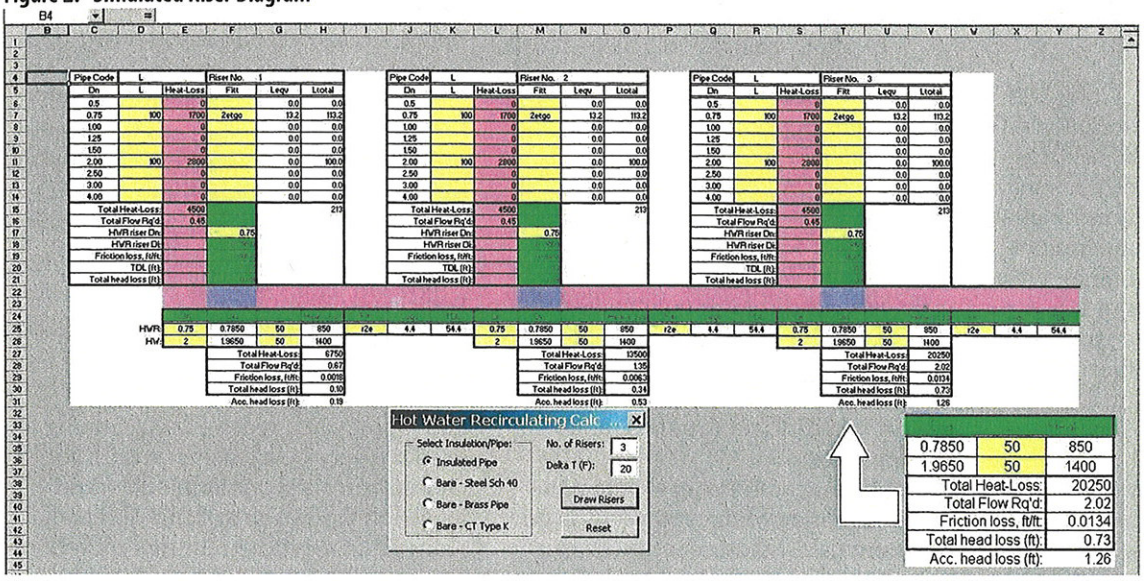
Range C15:F21 outputs the results, except for range C17:F17 where you enter the return pipe size in cell F17 (0.75, or three-fourths inches, in this case). Cell E15 summarizes the heat losses. Dividing cell E15 by a factor to obtain the flow in gallons per minute derives the total flow required in gpm, which is shown in cell E16. The factor is 10,000 for a temperature differential of 20°F, 5,000 for 10°F temperature differential, 2,500 for 5°F, and so on. Excel automatically enters the factor based on the delta T you

Table 1. Piping Heat Loss

(Btu/hr. Per Lineal Ft. For 140°F. Water Temp and 70°F. Room Temp.)				
		Bare Pipe		
Nominal Pipe Size	Insulated Pipe (½" Fiberglass)	Sched. 40 Steel	Brass, Copper, T.P.	Type K Copper
½"	15	35	26	19
¾"	17	43	32	26
1"	19	53	38	32
1¼"	21	65	46	39
1½"	25	73	53	46
2"	28	91	65	58
2½"	32	108	75	68
3"	38	129	90	81
4"	46	163	113	103
5"	55	199	138	127
6"	63	233	161	149
8"	80	299	201	188

Source: *Engineered Plumbing Design*, Alfred Steele, PE, CIPE

Figure 2. Simulated Riser Diagram



previously entered. (Note: 1 gal = 8.34 lb; 1 hr = 60 min; 1 Btu = 1 lb water at 1°F temperature change. Heat loss for 1 gpm flow at 1°F temperature drop = 8.34 × 60 = 500 (Btu/hr/°F/gpm). Therefore, required flow (gpm) = heat loss/500/delta T.)

Based on the pipe code in cell D4, the flow rate in cell E16, and the nominal riser size in cell F17, the actual inner diameter then appears in cell F18, and friction loss in feet/feet is calculated and displayed in cell F19. Cell H14 is the total riser length (the developed length plus equivalent length) of both supply and return piping. Because the return pipe length is roughly half of the total developed length, cell F20 shows the total length of the return piping ([100 + 100]/2 + 213 – [100 + 100] = 113). Cell F21 shows the total head loss within the riser section (in feet).

Range D24:K31 is for horizontal piping data and related calculation results. You input return and supply pipe sizes in cells E25 and E26 respectively, and cells F25 and F26 show actual inner sizes. You input developed lengths in cells G25 and G26 and fittings in cell I25. Cells H25 and H26 show heat losses, and cells J25 and K25 show the equivalent length and total length respectively.

The number in cell H27 is the total loss of both the risers and horizontal piping. Cell H28 shows the total flow required; cell 29 shows the friction loss in the horizontal piping; and cell H30 shows the total head loss of the horizontal piping. Cell H31 shows the total or accumulated head loss of the first section, including the risers and horizontal piping.

Drawing Riser Diagram. After inputting data in the template and selecting the pipe type, number of risers, and temperature differential in the UserForm (Insulated Pipe, three risers, and 20°F in this example), press the draw riser button. A simulated riser diagram appears on the screen (see **Figure 2**). The riser diagram shows the following:

- the total flow required for the whole system is 2.02 gpm (cell V28) and
- the total head loss for the whole system is 1.26 feet (cell V31).

The drawn risers are identical. You can input data according to actual size, length, and fittings, and the results will change accordingly. You don't need to recalculate the system. If you change the number of risers or temperature differential in the UserForm and press the draw riser button, the screen automatically resets.

Analysis

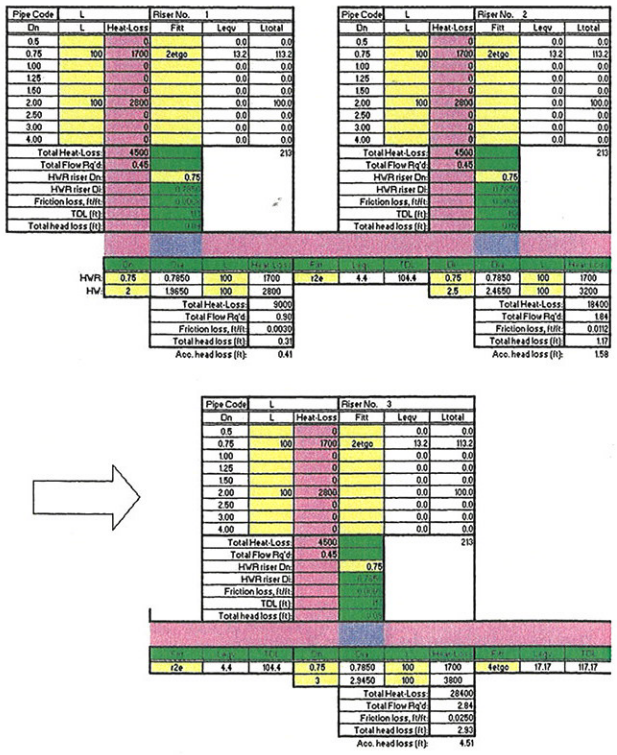
If you enlarge the horizontal sections from 50 feet to 100 feet and assume there are four elbows, one tee, one gate valve, and one check valve (4etgc) between the circulation pump and the last riser, the total heat loss is 28,400 Btu/hr, and the circulation flow required is 2.84 gpm. With the assigned pipe size and fittings the head loss will be 4.5 feet. (See **Figure 3**.)

If the temperature differential is set for 5°F, the flow will be 11.03 gpm and the head loss will be 15.7 feet by enlarging the horizontal recirculation piping from three-fourths inch to one inch.

In another example, consider a building that has 50 stories with four three-inch risers that are each 600 feet high, and the horizontal pipe between two adjoining risers is four inches in diameter and 150 feet long. The total heat loss then is 177,900 Btu/hr, and the circulation pump can deliver 17.8 gpm

...*continued on page* **27**

Figure 3. A Simplified Three-Riser Hot-Water Distribution System With Recirculation



...continued from page 25

at 20.1-foot head. The circulation piping sizes are from one inch to 1½ inch. (This riser diagram is not shown.) In this example, if the piping is copper tube type K without insulation, the total heat loss will be increased to 347,319 Btu/hr, and the pump shall have a flow of 34.7 gpm at 60.5-foot head if the circulation pipe sizes stay the same. But you easily can lower the head to 27.3 feet by increasing the circulation pipe diameter by one size.

The above-mentioned examples are simplified for analysis purpose. Hot-water recirculation is a closed loop system. Therefore, you don't need to consider elevation nor static losses.

Hot-water recirculation mostly is needed when few people are using the system and typically is least needed during peak hours, except in areas remote from the hot water source. Therefore, the head loss in the supply piping portion can be neglected. Some engineers prefer to double the friction loss in the circulation piping when sizing a circulator, which provides enough of a safety factor for the calculation.

Other than in very big buildings, recirculation pumps do not present a significant power consumption problem. A 1/6 recirculation pump can provide 15 gpm at 15 feet, which equals a 150-watt light bulb. Therefore, an aquastat may not be a critical component for the system.

You never should overlook the fact that a hot-water recirculation system is based on proper flow balancing. A hot-water recirculation system should not be oversized unnecessarily in order to save energy, whereas flow balancing and other unconsidered factors should be taken into account.

Field study is recommended to establish the relationship between adequate recirculation and energy savings. ■

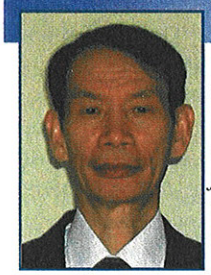
References

Engineered Plumbing Design II. Alfred Steele, revised by A. Calvin Laws, PE, CPD. ASPE, 2004.

"Hot-Water Recirculation Systems," *Plumbing Systems & Design.* Joseph V. Messina. ASPE. May/June 2004.

"Hot Water Circulation Systems," *ASPE Data Book, Volume 2,* Chapter 6: Domestic Hot Water Heating Systems. 1999.

Excel 2000 VBA, Programmer's Reference. John Green, Stephen Bullen, and Felipe Martins. Wrox Press Ltd. 2000.



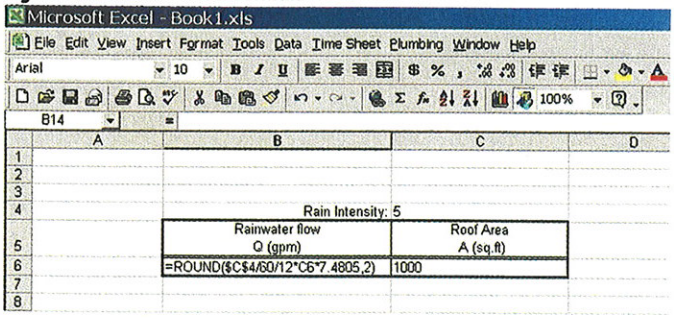
Anjian Lu, CPD, is an engineer with Vanderweil Engineers in Princeton, N.J. For further information, contact him at alu@vanderweil.com *or* luaj@yahoo.com.

Using Excel Functions in Plumbing Engineering

Anjian Lu, CPD

Plumbing engineers often must use formulas such as the Hazen-Williams Formula, Manning's Formula, and the Hardy Cross method to determine piping friction loss and flow capacity, check water pressure and size pumps, convert fixture unit to flow rate, calculate equivalent length, hot/cold water mixing, and fuel gas friction loss, among many other calculations.

Figure 1. Rainwater Flow Formula



Before the advent of the computer era, scientists and engineers found laws or relations between relevant factors, expressed those relations with formulas, and established many tables, charts, and other aids for use in daily engineering tasks. But these charts are not always practicable. For example, to determine a pipe's friction loss you can consult a thick manual containing different pipe sizes and flow rate increments. But the manual only provides one Hazen-Williams C factor of 100 because a table incorporating small increments of every parameter would be too big and clumsy for use.

Luckily, today you can perform all of these calculations using computer programs. Microsoft Excel is one of the most powerful software programs for this purpose because of its inherent functions, User-Defined Functions (UDFs), and other capabilities.

Functions are the fundamental components of computer-aided computations. Many functions come pre-loaded on Excel, and users can write functions based on their own needs. Functions can be very simple procedures or quite long programs. In this article I use a simple function as an example, but the concepts work for longer functions as well.

Excel's Inherent Functions

In Excel 2000 and earlier versions, the Paste Function icon (*fx*) is located on the standard toolbar. Clicking this icon brings up a form containing common functions. Excel 2002 and 2003 do not have the Paste Function icon on the standard toolbar. You can click the Σ icon on the standard toolbar to enter the SUM() function or click the arrow beside it to show the pull-down menu with other functions. If you select the last item (More Functions), a form called Insert Function appears. You can either type a brief description of what you want to do and then click Go or select a category in the second form to show more functions. If you have UDFs, the last category will be User Defined. All of these are self-explanatory.

If you use Excel 2000 or earlier versions, in the Paste Function form, a function's variables and an explanation appear in the lower left corner when that function is selected. Whether in a cell or in the formula bar at the top of the workbook, functions should be preceded by an equal sign. Otherwise, Excel treats the entered numbers and functions as a text string. You can use as many levels of functions as you wish, such as: =ROUND(ABS(SIN(PI()/4)),2). This formula returns the value 0.71.

If you use Excel 2002 or 2003 and select a function from the Insert Function form and click OK, a Function Arguments form appears. You then can enter data for each argument and get the result directly. (As plumbing engineers, we use the term *variable* instead of *argument*.) But in most cases you can enter a cell address instead of a number, so the function reacts as the value in a cell is changing.

Entering a Formula

Let's calculate rainwater flow as an example of entering formulas. As rainwater flow is the product of rain intensity and collection area, the calculation is as follows:

$$Q = H/(60 \times 12) \times A \times 7.4805$$

where

Q = rainwater flow (gallons per minute)
H = rain intensity (inches/hour)
A = roof area (square feet)
60, 12, and 7.4805 = factors to convert the final unit to gpm

If the rain intensity is five inches/hour and the roof area is 1,000 square feet, and we enter them in cells C4 and C6 respectively, then the Excel function is:

=ROUND(C4/60/12*C6*7.4805,2)

ROUND is an inherent function, which has the format ROUND(number, digit). The first argument, number, is a number to be rounded and the second, digit, is for specifying the rounded number's decimal points.

You may enter the formula in the formula bar or directly in cell B6 (see Figure 1). Hit the enter key or move the cursor out of cell B6, and the value 51.95 appears. Figure 1 is in Formula

Figure 2. Rainwater Flow Calculation Sheet

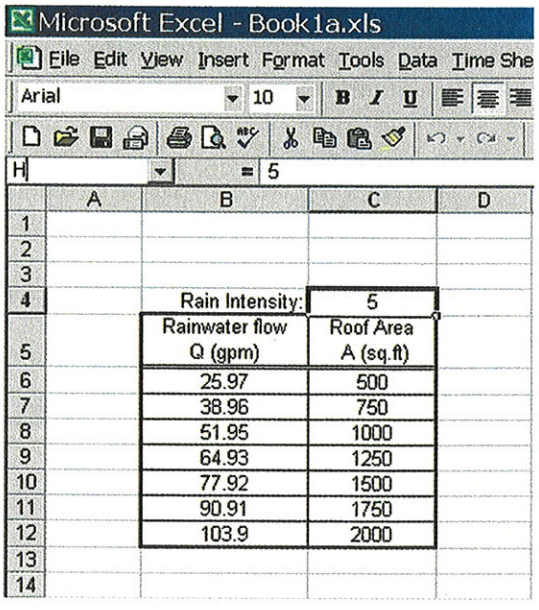
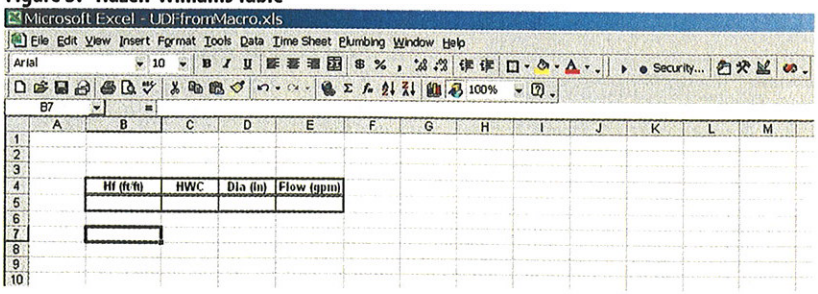


Figure 3. Hazen-Williams Table



view (on the Worksheet Menu Bar follow Tools→Options and check Formulas under View options). The dollar sign keeps the cell fixed in the same position when using the Copy and Paste functions.

Now you can enter other roof areas in column C and copy the formula in column B without needing to calculate one flow rate after another.

Using Names

Using too many cell addresses, especially for functions with more than three variables, may be confusing. The solution is to label cells with particular names. For example, using the previous example, select cell C4. In the name box to the left of the formula bar, change C4 to H and hit enter. Now cell C4 is named H and the formula becomes the following:

=ROUND(7.4805/60/12*H*C6,2) or
=ROUND(0.01039*H*C6,2)

This formula is more meaningful with H representing rain intensity, which is a constant in a given location. Do not replace cell C6 with a name because roof area is a variable.

Figure 2 shows a simple calculation sheet using the above formula. Name cell C4 as H and enter roof areas in cells C6 through C12. Enter the formula in cell B6, copy and paste it in cells B7 through B12, and the flow rate values appear instantly.

Using Macro Recorder

Excel's macro recorder can record the work done over a workbook or worksheet. Macros are actually procedures in Visual Basic automatically created by Excel.

Follow these steps to record a macro:

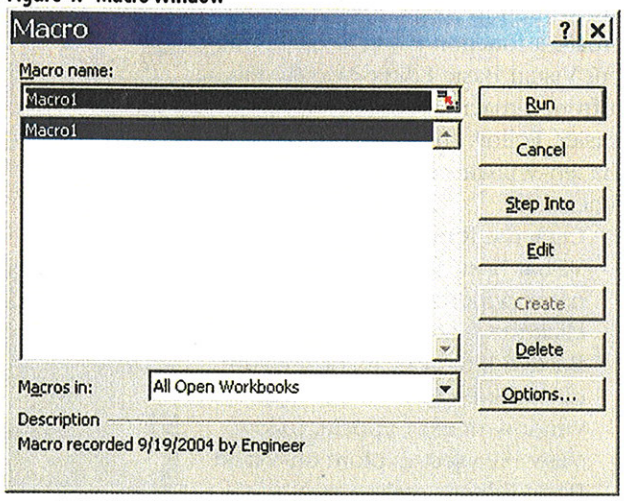
1. Create a new workbook and save it as UDFfromMacro.xls.
2. Create a table on sheet 1 as shown in Figure 3.
3. Bring the Visual Basic toolbar to view by following View→Toolbar→Visual Basic on the Worksheet Menu Bar. The solid circle is the Record Macro tool and the triangle is the Run Macro tool.
4. Click the solid circle and the Record Macro window pops up. You may change the macro name, storage place, and description if you like. In this example we leave the name as Macro1. Click OK to start recording Macro1. The solid circle in the Visual Basic toolbar is now a solid square, or Stop Recording.
5. Name cell B5 as S and enter the following formula:

   =10.4315*(Q/Ch)^1.85/D^4.8655
6. The formula appears in the formula bar. #DIV/0! appears in cell B5 but do not be concerned with that for now. Name cell C5 as Ch, cell D5 as D, and cell E5 as Q. (Ch stands for the Hazen-Williams Coefficient because C already is preserved by Excel.)
7. Click the solid square icon to stop recording. The icon changes back to a solid circle. Clear the contents in cells B5 through E5 and save and close the workbook.

Once you have recorded a macro, you can use it again and again. Simply open the workbook UDFfromMacro.xls and click the Run Macro icon (the solid triangle) on the Visual Basic toolbar. In the Macro window that appears, select Macro1 and click the Run button (see Figure 4).

The Hazen-Williams Formula should be in cell B5. Type 100, 4, and 100 in cells C5, D5, and E5 respectively.

Figure 4. Macro Window



Cell B5 now should show the value 0.012275076 (see Figure 5). This is the friction loss in feet/feet when Ch = 100, D = 4, and Q = 100. You can type any other reasonable numbers in cells C5 through E5, and a new friction loss will appear in cell B5. If for some reason the formula in cell B5 is erased, run Macro1 again and follow the above-mentioned steps to restart the calculation.

Figure 5. Calculated Hazen-Williams Formula

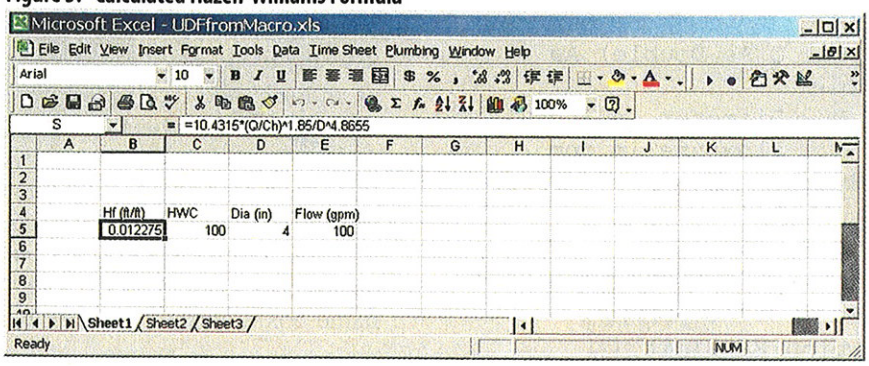
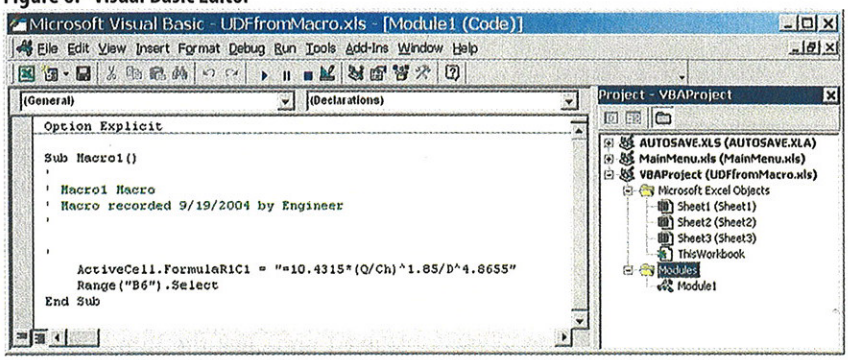


Figure 6. Visual Basic Editor



Creating UDFs from Macros

In addition to the pre-loaded functions, users can create their own functions. Both macros and functions are written in Visual Basic. You can create a function using Excel's inherent Visual Basic Editor or converting it from a macro. The latter method is easier. Follow these steps to convert the Hazen-Williams Formula function from Macro1:

1. Click the Run Macro tool on the Visual Basic toolbar. In the Macro name window, select Macro1 and click the Edit button. The Visual Basic Editor appears (see Figure 6). If the Project – VBAProject window doesn't appear, follow View→Project Explore on Visual Basic Editor's main menu.
2. Right click on VBAProject (UDFfromMacro.xls). A pop-up menu appears. Move the cursor to Insert and select Module. A new module, Module2, is added under Module1.
3. Copy all the contents in Macro1 and paste them in Module2. If two Option Explicit lines appear, delete one of them. Revise the pasted contents to read as follows:

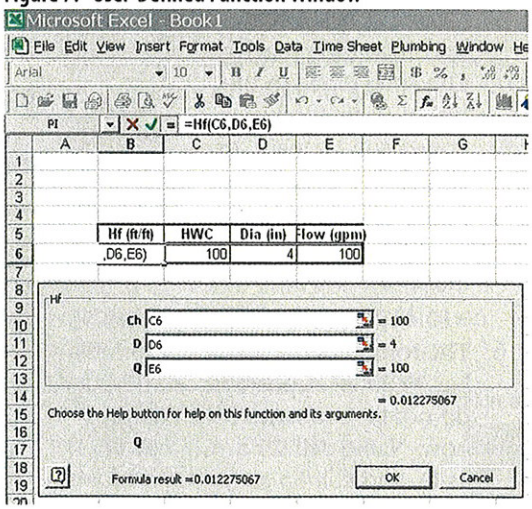
```
Option Explicit
Function Hf(ByVal Ch As
Double, ByVal D As Double, _
ByVal Q As Double) As
Double
' D - Diameter in inch
' Q - Flowrate in gpm
Hf=10.4315*(Q/Ch)^1.85/
D^4.8655
End Function
```

4. If you use the metric system, revise the function as follows:

```
Option Explicit
Function Hf(ByVal Ch As
Double, ByVal D As Double, _
ByVal Q As Double) As
Double
' D - Diameter in mm
' Q - Flowrate in L/s
Hf=1.18478*10^10*(Q/Ch)^1.85/
D^4.8655
End Function
```

Save the UDFfromMacro.xls file. You can rename Module2 by selecting the Properties Window from View on Visual Basic Editor's main menu.

Figure 7. User-Defined Function Window



You cannot use the UDF you just created quite yet. Follow these two steps to make the UDF available for use in other workbooks.

1. Save the workbook UDFfromMacro as an Excel Add-In file. Select Save As from the Worksheet Menu Bar. Select Microsoft Excel Add-In (*.xla) from the Save as type: drop-down box at the bottom of the Save As window. The file name UDFfromMacro.xla should be in the File name box. Choose the folder in which you want to save the UDF and click Save.
2. Add UDFfromMacro.xla to the Add-Ins list. From the Worksheet Menu Bar follow Tools→Add-Ins. The Add-Ins window appears. Click the Browse button to find the file you just saved and click OK. If you check the Add-Ins window again you see that a new item, UDFfromMacro, is added to the list and that the box to its left is checked. Now the Hazen-Williams Formula is loaded in Excel, and you can use it in any worksheet.

Putting UDFs to Work

In a new Excel workbook, bring the Paste Function on Insert Function window on screen as previously described. Click User Defined in the Function category box. In the Function name box select Hf, which is the function we just created, and click OK. The UDF appears as shown in Figure 7 when using Excel 2000 or earlier version. Version 2002 and 2003 redesigned the form and added a title for it as Function Arguments.

Link the argument boxes to the related cells (Ch→C6, D→D6, Q→E6) and click OK. The friction loss (Hf) in feet/feet is shown in the selected cell. In this example it is 0.012275 when C6 = 100, D6 = 4, and E6 = 100. If you know the function's name, you can type it directly in a cell preceded by an equal sign. You also can type values for the arguments in the parentheses, such as =Hf(100,4,100).

Functions are very important because they are the fundamental components of computer-aided computations. Using these guidelines you can create more UDFs to aid in plumbing engineering calculations. ■

Anjian Lu is an engineer with Vanderweil – NCR in Alexandria, Va. For further information, contact him at **luaj@yahoo.com**.

Creating and Using a Fixture Unit Versus Flow Calculator

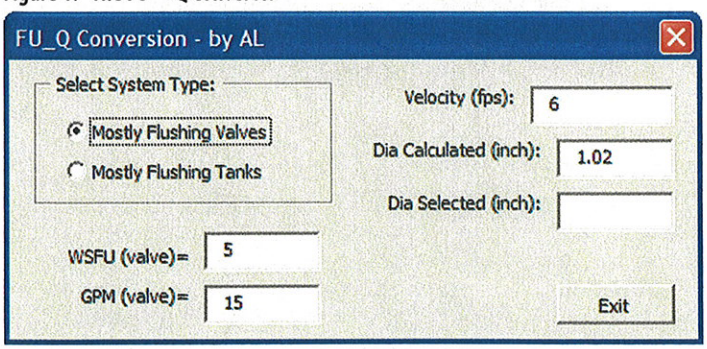
Anjian Lu, CPD

The Hunter's Curve Method is the basis for sizing water supply systems. However, calculating a flow (Q) based on a water supply fixture unit (WSFU) to size numerous pipes in a plan or a riser diagram can be tedious. The WSFU→Q conversion calculator discussed in this column, created using Microsoft Excel's Visual Basic Application (VBA) feature, can resolve this problem.

The WSFU→Q Calculator

The calculator (see Figure 1) offers two system type options: mostly flushing valves and mostly flushing tanks. When the user inputs the WSFU in the first text box on the left, the flow in gallons per minute shows in the text box below. The first text box on the right side of the calculator shows the default velocity, and the second box outputs the calculated diameter based on the preset velocity of six feet per second. In the third box the user can select a diameter based on actual conditions. Clicking the exit button terminates the calculation, as does clicking the red X in the upper right corner.

Figure 1. The FU→Q Converter



To use the calculator, first you need to create a database of the WSFUs and flows from the Hunter's Curve sizing guide. To do so, we are going to use Excel's range and name scheme and then assign data to the variables.

Create a workbook (named Topic 2 in this example) and save it in a directory. The workbook will contain three worksheets by default. In sheet 3, type the WSFU and flow data as shown in Figure 2. Highlight range D9:E50 and name it rgFUv, and highlight range B5:C50 and name it rgFUt. To name the range, you need only to type the name in the Name Box in the upper left corner, right above the column header row (A, B, C, …).

Creating the User Form and Controls

In Excel, the form shown in Figure 1 is called the UserForm and the frames, boxes, buttons, and labels are called controls.

When you click the Visual Basic Editor icon on the Visual Basic toolbar, the screen converts to Microsoft Visual Basic. Figure 3 shows the icon location on the toolbar. (Note: I am using Excel 2003 in this article. Icons and screens may look slightly different in other versions.) Highlight VBAProject (Topic 2) in the Project Explorer window on the left side of the screen. (If the window did not appear automatically, click View in the main menu and select Project Explorer from the pull-down menu.) Then select UserForm from the Insert menu. A form called UserForm1 and a toolbox should appear on the screen (see Figure 4). If a toolbox does not show, click the Control Toolbox icon on the Standard toolbar.

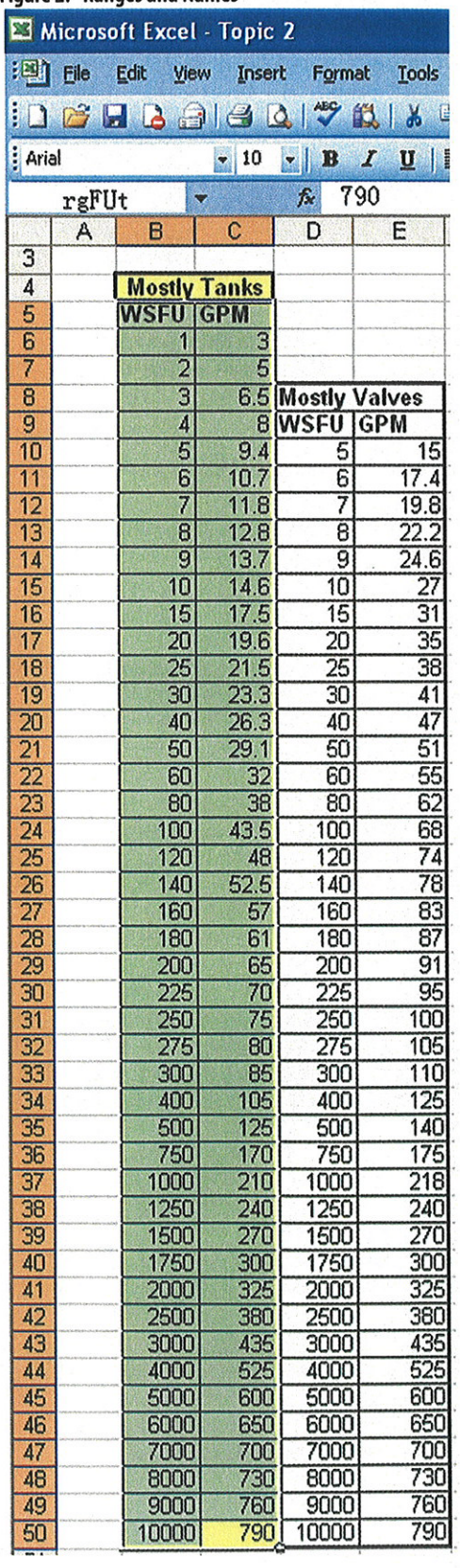
Click on Userform1 and then select Properties window from the View pull-down menu. The Properties window of Userform1 will show on the screen (see Figure 5). Rename UserForm1 as frmFUQconvert and the caption as FU_Q Conversion. From the Toolbox menu, drag and drop controls on the form. Rename them as listed in Table 1. The UserForm now looks like that shown in Figure 6.

Writing Codes for the Controls

Now you need to write codes for the controls to make them work.

frmFUQconvert. Double clicking on the form leads you to the code

Figure 2. Ranges and Names



	A	B	C	D	E
3					
4		Mostly Tanks			
5		WSFU	GPM		
6		1	3		
7		2	5		
8		3	6.5	Mostly Valves	
9		4	8	WSFU	GPM
10		5	9.4	5	15
11		6	10.7	6	17.4
12		7	11.8	7	19.8
13		8	12.8	8	22.2
14		9	13.7	9	24.6
15		10	14.6	10	27
16		15	17.5	15	31
17		20	19.6	20	35
18		25	21.5	25	38
19		30	23.3	30	41
20		40	26.3	40	47
21		50	29.1	50	51
22		60	32	60	55
23		80	38	80	62
24		100	43.5	100	68
25		120	48	120	74
26		140	52.5	140	78
27		160	57	160	83
28		180	61	180	87
29		200	65	200	91
30		225	70	225	95
31		250	75	250	100
32		275	80	275	105
33		300	85	300	110
34		400	105	400	125
35		500	125	500	140
36		750	170	750	175
37		1000	210	1000	218
38		1250	240	1250	240
39		1500	270	1500	270
40		1750	300	1750	300
41		2000	325	2000	325
42		2500	380	2500	380
43		3000	435	3000	435
44		4000	525	4000	525
45		5000	600	5000	600
46		6000	650	6000	650
47		7000	700	7000	700
48		8000	730	8000	730
49		9000	760	9000	760
50		10000	790	10000	790

Table 1. The Controls

Control	Name	Caption	Value
UserForm	frmFUQconvert	FU→Q Conversion	
Frame	fraSelectSys	Select System Type	
OptionButton	optValve	Mostly Flushing Valves	True
OptionButton	optTank	Mostly Flushing Tanks	False
Label	lblInput	(Blank)	
Label	lblResult	(Blank)	
Label	lblVelocity	Velocity (fps)	
Label	lblDiameter	Dia Calculated (inch)	
Label	lblDiaSelect	Dia Selected (inch)	
TextBox	txtInput		
TextBox	txtResult		
TextBox	txtVelocity		6
TextBox	txtDiameter		
TextBox	txtDiaSelected		
CommandButton	cmdExit	Exit	

section and a subprocedure with only two lines. First, write the following sentences above the first line.

```
Option Explicit
Const pi = 3.1415926535

Private Sub UserForm_
Initialize()

End Sub
```

Then enter codes above the last line, so the whole paragraph reads like this:

```
Option Explicit
Const pi = 3.1415926535

Private Sub UserForm_
Initialize()
  optValve.Value = True
  optTank.Value = False
  lblResult.Caption = "GPM=
"
  lblInput.Caption = "WSFU
(valve)= "
  txtResult.Locked = True
  txtInput.Text = 5
End Sub
```

Only one Option Explicit line should be listed at the beginning, or you will get an error message when running the program. The second sentence defines a constant pi for calculating the circular pipe sectional area. The fourth line tells Excel that the mostly flushing valves option will be our default choice. Line seven shows the label caption for the mostly flushing valves choice. Line eight locks the txtResult output box. Line nine is for setting the WSFU, which you can change as required.

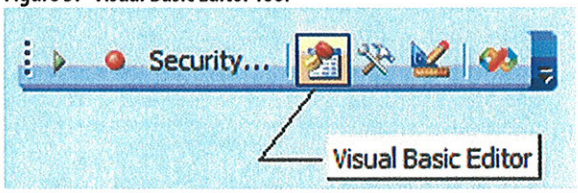
optValve and optTank. Double click the optValve option control and write one sentence, GetGPM, between the two lines automatically created. (Note: There are other ways to bring up the code page. You can select Code from the View pull-down menu. You also can select View Code from the Project Explore window, which you bring up by selecting Project Explore from View on the main menu.)

```
Private Sub
optValve_Click()
  GetGPM
End Sub
```

Do the same for optTank.

```
Private Sub optTank_Click()
  GetGPM
End Sub
```

Figure 3. Visual Basic Editor Tool



GetGPM is a subprocedure that calculates the flow based on the inputted WSFU. It will be shared by other controls. (Creating the GetGPM subprocedure is discussed later.)

txtInput. Similarly, double click the txtInput text box. Two lines are created automatically. Add GetGPM between them.

```
Private Sub txtInput_
Change()
  GetGPM
End Sub
```

txtResult and txtVelocity. Double click on the txtResult and txtVelocity text boxes and enter CalcDiameter as follows.

```
Private Sub txtResult_
Change()
  CalcDiameter
End Sub

Private Sub txtVelocity_
Change()
  CalcDiameter
End Sub
```

Notice that the first sentence of the last three subprocedures ends with Change() instead of Click(). This tells Excel VBA to respond to any inputted change.

txtDiaSelected. Double click the txtDiaSelected text box and write the following codes between the two lines created automatically.

```
Private Sub txtDiaSelected_
Change()
  Dim Q As Double
  Dim V As Double
  Dim D As Double

  On Error GoTo ErrorTrap
  D = txtDiaSelected.Text
  D = D / 12
  Q = txtResult.Text
  Q = Q * 0.002228
  V = (4 * Q / (pi * D ^ 2))
  txtVelocity.Text =
WorksheetFunction.RoundUp(V,
2)
  Exit Sub
ErrorTrap:

End Sub
```

In this subprocedure we define variables for holding the diameter (D) and flow (Q) (shown in the output txtResult.Text text box), as well as the velocity (V).

Figure 4. Visual Basic Editor Screen

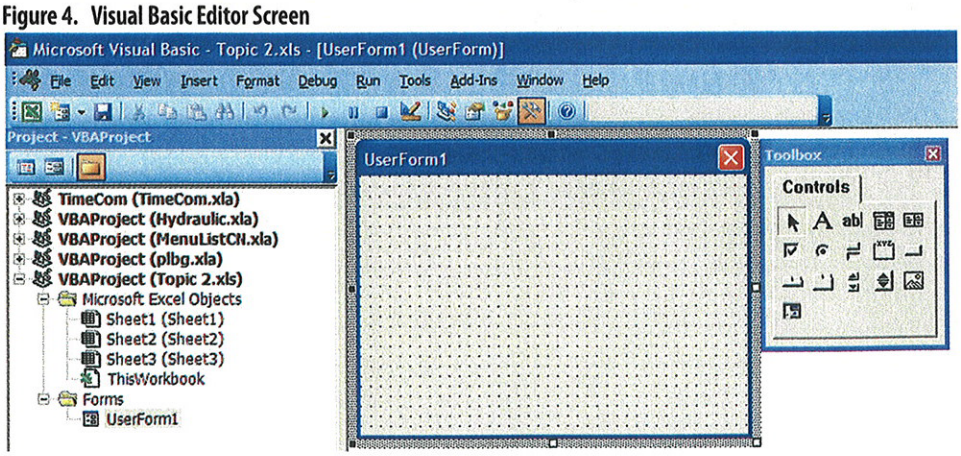
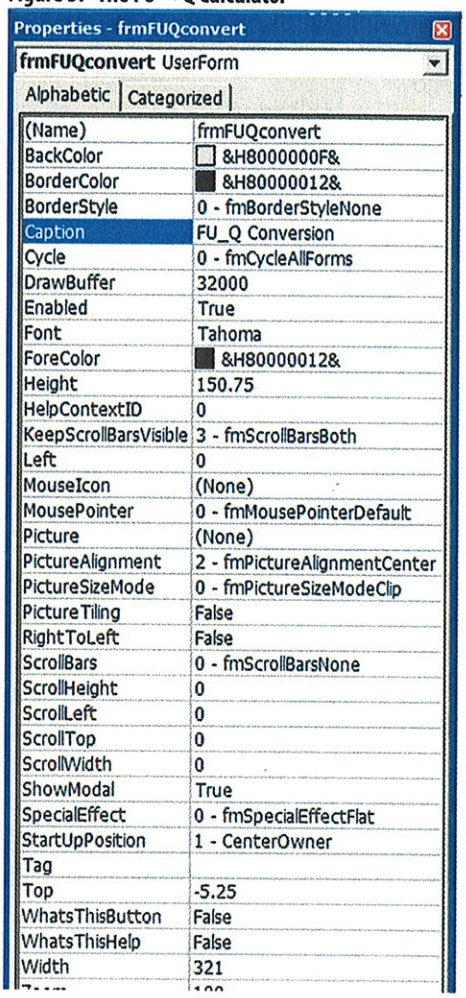


Figure 5. The FU→Q Calculator



Line eight converts the inputted diameter from inches to feet. The calculated velocity to two decimal points then shows in the text box.

The sentence On Error GoTo ErrorTrap is very important because it will trap any input error in the txtDiaSelected text box to avoid a data type mismatch, which could cause the program to crash. Be sure to add the sentence Exit Sub before ErrorTrap:.

cmdExit. You can write a command to end the program. It is simplest to write the word End.

```
Private Sub cmdExit_Click()
  End
End Sub
```

Writing Common Procedures

As mentioned above, GetGPM is a common subprocedure. Let's add the following:

```
Private Sub GetGPM()
  Dim dGPM As Double

  txtVelocity.Text = 6
  txtDiaSelected.Text = ""
  If optValve.Value = True
Then
    lblInput.Caption = "WSFU
(valve)= "
    lblResult.Caption = "GPM
(valve)= "

    On Error GoTo ErrorTrap
    dGPM = QvFU(txtInput.
Text)
    txtResult.Text =
WorksheetFunction.
Round(dGPM, 1)
  Else
    lblInput.Caption = "WSFU
(tank)= "
    lblResult.Caption = "GPM
(tank)= "

    On Error GoTo ErrorTrap
    dGPM = QtFU(txtInput.
Text)
    txtResult.Text =
WorksheetFunction.
Round(dGPM, 1)
  End If
  Exit Sub

ErrorTrap:
  txtInput.Text = 5
  MsgBox "Please enter a
number", vbCritical
End Sub
```

In this subprocedure you define the variable dGPM for holding the flow calculated by the functions QvFUO and QtFUO. Line three defines the velocity at six fps, which you can change based on the selected diameter. Line four clears the diameter selected in the previous operation. The If – Else – End If condition procedure identifies the choice you selected. It will change the labels for input and output text boxes accordingly and calculate flow in gpm based on either the QvGPM or QtGPM function. Lines eight and 14 are for catching type mismatch errors. If an error occurs, the error trap will set the inputted WSFU at five, and the warning "Please enter a number" will appear.

To suggest a diameter based on a six-fps flow velocity, you write the following subprocedure:

```
Private Sub CalcDiameter()
  Dim Q As Double
  Dim V As Double
  Dim D As Double

  On Error GoTo ErrorTrap
  V = txtVelocity.Text
  Q = txtResult.Text
  Q = Q * 0.002228
  D = (4 * Q / (pi * V)) ^
0.5 * 12
  txtDiameter.Text =
WorksheetFunction.RoundUp(D,
2)

  Exit Sub
ErrorTrap:

End Sub
```

Writing Functions

Functions written by users are called User Defined Functions (UDFs). The difference between a subprocedure and a function is that the latter returns a value. For the program to return the flow in gpm when you input a WSFU, you need to write a UDF for the mostly flushing valves water supply system as follows.

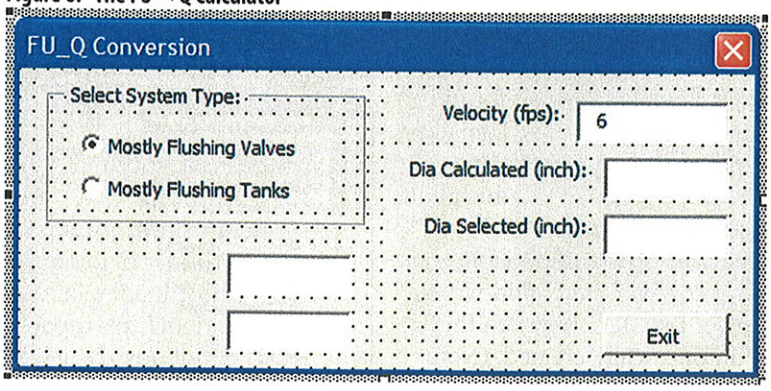
```
Function QvFU(SFUvalve As
Single) As Single

  Dim vaSFUv As Variant
  Dim I As Integer

  vaSFUv = ThisWorkbook.
Sheets(3).Range("rgFUv")

  For I = LBound(vaSFUv) + 2
To UBound(vaSFUv)
```

Figure 6. The FU→Q Calculator



```
    If SFUvalve = 10000 Then
QvFU = 790
    If SFUvalve < 5 And
SFUvalve <> 0 Then QvFU = 15
    If (vaSFUv(I, 1) >
SFUvalve And vaSFUv(I - 1,
1) <= SFUvalve) Then
      QvFU = vaSFUv(I
- 1, 2) + (vaSFUv(I, 2) -
vaSFUv(I - 1, 2)) * _
      (SFUvalve - vaSFUv(I
- 1, 1)) / (vaSFUv(I, 1) -
vaSFUv(I - 1, 1))
      Exit Function
    End If
  Next I
End Function
```

The data type of the argument of SFUvalve in function QvFU() is defined as single for accepting the WSFU passed. In the procedure are two variables: vaSFUv and I. The former is a variant for holding the data in range rgFU on sheet 3. vaSFUv is a two-dimensional data type. The first dimension holds WSFUs and the second holds flows, including their headers. Line four assigns Range("rgFUv") to vaSFU. The following For … Next loop checks the inputted WSFU and tries to match the data in vaSFU.

The LBound(vaSFUv) value is 1, which corresponds to the head row; since we want to start from the second row, we set the first I at 3 (i.e., I = LBound(vaSFUv) + 2). Line six is for the last WSFU (i.e., if WSFU=10,000; Q=790 GPM). Line seven sets Q=15 gpm when the WSFU is less than five but not zero. The following If — End If condition section locates the inputted WSFU between two numbers in the array. We use interpolation to calculate the flow in gpm. The calculation ends after this, and the value returns to the calling sentence.

Similarly, for the system with mostly flushing tanks we write the following function:

```
Function
QvFU(SFUvalve As
Single) As Single
  Dim vaSFUv As Variant
  Dim I As Integer
  vaSFUv = ThisWorkbook.
Sheets(3).Range("rgFUv")
  For I = LBound(vaSFUv) + 2
To UBound(vaSFUv)
    If SFUvalve = 10000 Then
QvFU = 790
    If SFUvalve < 5 And
SFUvalve <> 0 Then QvFU = 15
    If (vaSFUv(I, 1) >
SFUvalve And vaSFUv(I - 1,
1) <= SFUvalve) Then
      QvFU = vaSFUv(I
- 1, 2) + (vaSFUv(I, 2) -
vaSFUv(I - 1, 2)) * _
      (SFUvalve - vaSFUv(I
- 1, 1)) / (vaSFUv(I, 1) -
vaSFUv(I - 1, 1))
      Exit Function
    End If
Next I
End Function
```

Writing a Button on the Worksheet

Writing a button on the worksheet is similar to that on the UserForm. Switch the worksheet from sheet 3 to sheet 1. Select the Control Toolbox icon next to the Visual Basic Editor icon (see Figure 3). The Design Mode icon is checked automatically. Select the command button icon in the Control Toolbox and draw a box on the worksheet to place it. Right click the button and select Properties from the pop-up menu. Change the button name from CommandButton1 to cmdShowForm and its caption to Show Form on the properties window (see Figure 7).

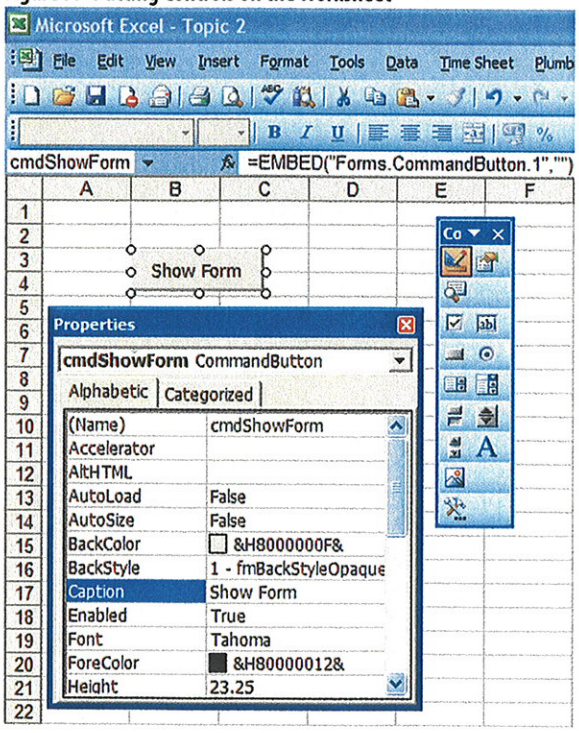
Double click the Show Form button and write the following procedure between the two lines automatically created.

```
Private Sub cmdShowForm_
Click()
  frmFUQconvert.Show
vbModeless
End Sub
```

The keyword vbModeless enables you to work on the worksheet with the User Form shown on the screen. Go back to sheet 1 and click the Exit Design Mode icon on the Visual Basic toolbar.

Be sure to save the workbook. Now when you need to calculate the flow from WSFU, simply open the workbook and click on the Show Form button on sheet 1. The calculator will show up ready for use. ■

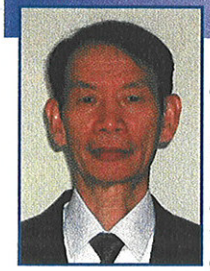
Figure 7. Putting Controls on the Worksheet



Recommended Reading

Green, John, Stephen Bullen and Felipe Martins. *Excel 2000 VBA, Programmer's Reference*. Wrox Press Ltd., 2000.

Green, John, Paul T. Kimmel, Stephen Bullen, Rob Bovey, et al. *Excel 2003 VBA Programmer's Reference*. Wrox Press Ltd., 2004.



Anjian Lu is an engineer with Vanderweil – NCR in Alexandria, Va. To comment on this article, or for more information, contact **articles@psdmagazine.org**.

Creating Custom Menus to Organize Programs in Excel

Anjian Lu, CPD

After you have accumulated several reusable Excel templates, Add-Ins, UserForms, and other programs, you may want to organize them for convenient use. Fortunately, Excel has a feature to help you do this.

You can create custom menus on the main menu toolbar to hold your plumbing calculation tools. Figure 1 shows two custom menus: Time Sheet and Plumbing. Under the Plumbing menu are several submenus. You can create as many submenus and sub-submenus as you need. For example, Figure 1 shows the Water Demand submenu's three sub-submenus: Max Daily/Hourly, Fixture Unit, and FU_Flow Conversion. The latter is the calculator featured in this column in *PSD*'s May/June 2005 issue.

Now let's discuss how to add custom menus to the existing main menu bar, using the Plumbing menu as an example.

Writing the Code

To add a custom menu, you first need to write a subprocedure, or subroutine.

Open the file you created for converting water supply fixture unit (WSFU) to flow from the last issue's column. Save this file as an Excel Add-In, naming it PlbgMenu.xla, in the directory C:\PlumAl\Addins. Click the Visual Basic Editor (VBE) icon on the Visual Basic toolbar to bring up the Visual Basic window. In the Project — VBAProject window you will see the file you just saved, named VBAProject (PlbgMenu), in the directory tree. Highlight the file. Select Module from Insert on the main menu bar or right click on the highlighted file and select Insert — Module. The VBE window will appear automatically with this caption: Microsoft Visual Basic — PlbgMenu — [Module1 (Code)]. Change the Module1 default name to modAddPlbgMenu in its Property window. Now the Project Explore window will be similar to the one shown in Figure 2. (Based on the contents of your Excel application, the directory tree may look different.)

Write the subroutine found in the sidebar and name it AddPlbgMenu(). The first line creates an object variable cbMSMenuBar as a CommandBar type. The second line creates a control as type CommandBarControl (the Plumbing menu). The third line creates a variable to hold the index (or position) of an existing menu. Line four defines the command bar. The fifth line gets the index of the existing Window menu. This variable, iWindowIndex, will be used to define the position of the Plumbing menu. Line six places the Plumbing menu before the Window menu on the main menu bar.

The With muPlumbing — End With section holds all the tools you have created. The first line sets the menu name as Plumbing on the main menu bar. The embedded With — End With section adds an item to the drop-down menu, in this case Begin New Project. The sentence .OnAction = "ToBeDone" tells Excel to call the subroutine ToBeDone. You put a prime (') before this sentence to make it a comment, so the program won't execute it.

Figure 1. The Custom Menus

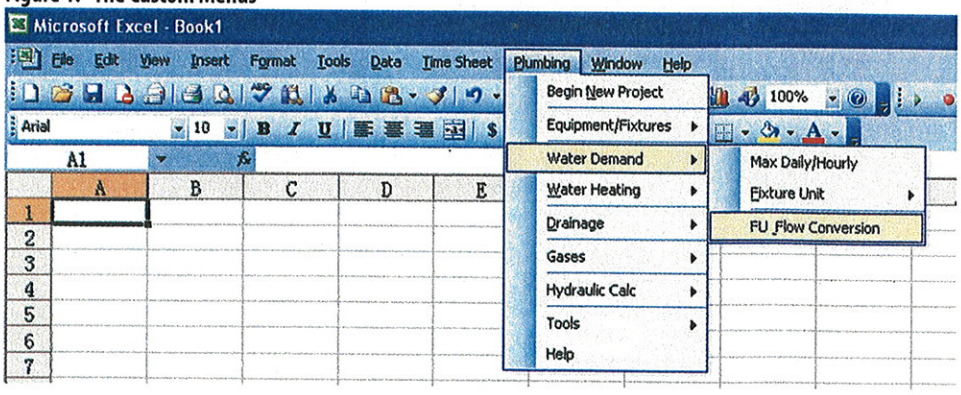
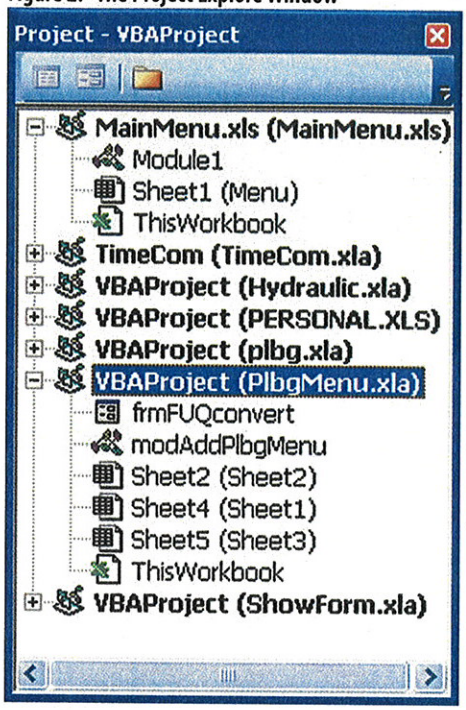


Figure 2. The Project Explore Window



Now let's look at the third embedded With — End With section. Within this section are three embedded With — End With sections. The first and second lines add the Water Demand submenu to the drop-down menu list. The .BeginGroup = True line in the first embedded section places a line on the main menu bar between the different menus. These three With — End With sections add the Water Demand submenu's sub-submenus: Max Daily/Hourly, Fixture Unit, and FU_Flow Conversion.

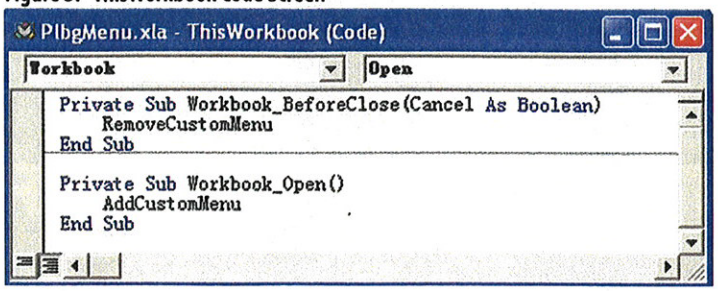
Writing Responding Subroutines

The menu's added control has a property called OnAction. You need to write subroutines to respond to it. To bring up the fixture unit versus flow calculator form, write the following subroutine:

```
Sub ShowfmFUQconvert()
   frmFUQconvert.Show
vbModeless
End Sub
```

The keyword vbModeless enables you to do other work without closing the form.

Figure 3. ThisWorkbook Code Screen



Similarly, you can include the hot water circulation system sizing template introduced in the November/December 2004 issue by writing the following subroutine:

```
Sub HWRecirculation()
   Workbooks.Add Template:=
"C:\PlumAl\WorkSheets\
HWRcalc.xlt"
End Sub
```

This subroutine uses the file HWRcalc.xlt as the template and creates a worksheet named HWRcalc1.xls.

To remind yourself and other users about menus that need to be developed, write the following subroutine:

```
Public Sub ToBeDone()
   MsgBox "To Be Developed",
vbInformation, "PlumAl"
End Sub
```

Code for Removing the Menu

To delete the Plumbing menu, write the following subroutine:

```
Public Sub RemovePlbgMenu()
  Dim cbWSMenuBar As
CommandBar
  On Error Resume Next
  Set cbWSMenuBar =
CommandBars("Worksheet Menu
Bar")
  cbWSMenuBar.
Controls("Plumbing").Delete
End Sub
```

This is the counterpart to the subroutine previously created. The line On Error Resume Next handles errors. Without this sentence, an error message will be thrown out and the program will crash if the Plumbing menu doesn't exist. With it, the program will bypass the error and execute to the end.

Adding and Removing a Menu

The last step is to add the Plumbing menu to the main menu bar and remove it when the PlbgMenu.xla Add-In file is excluded from the Add-In menu.

In the Project Explore window there is a file called ThisWorkbook (see Figure 2). Double click the file to bring up the code screen (see Figure 3).

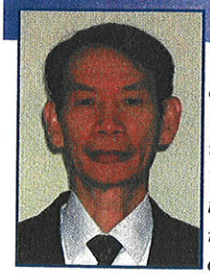
Choose Workbook from the left box on the top and choose Open from the right box. A two-line subroutine will be created automatically. Add a line between the two lines, so the whole subroutine reads as follows:

```
Private Sub Workbook_Open()
   AddPlbgMenu
End Sub
```

Then choose BeforeClose from the right box and write a line between the two lines automatically created, so the whole subroutine reads as follows:

```
Private Sub Workbook_
BeforeClose(Cancel As
Boolean)
   RemovePlbgMenu
End Sub
```

Save this file and go to the Excel Worksheet screen. Add the file to the list in the Add-Ins form by selecting Add-Ins ... from Tool on the main menu. The Plumbing menu will be created and inserted automatically before the Window menu on the main menu bar. ■



Anjian Lu is an engineer with Vanderweil – NCR in Alexandria, Va. To comment on this article, or for more information, contact articles@psdmagazine.org.

The AddPlbgMenu Subroutine Code

```
Option Explicit
Public Sub AddPlbgMenu()
 Dim cbWSMenuBar As CommandBar
 Dim muPlumbing As CommandBarControl
 Dim iWindowIndex As Integer
 Set cbWSMenuBar = CommandBars("Worksheet Menu Bar")
 iWindowIndex = cbWSMenuBar.Controls("Window").Index
 Set muPlumbing = cbWSMenuBar.Controls.Add(Type:= _
     msoControlPopup, Before:=iWindowIndex)
 With muPlumbing
   .Caption = "&Plumbing"
   With .Controls.Add(Type:=msoControlButton)
    .Caption = "Begin &New Project"
'.OnAction = "ToBeDone"
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .Caption = "Equipment/Fixtures"
    .BeginGroup = True
'.OnAction = "ToBeDone"
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .Caption = "Water Demand"
    .BeginGroup = True
    With .Controls.Add(Type:=msoControlButton)
     .BeginGroup = True
     .Caption = "Max Daily/Hourly"
'.OnAction = "ToBeDone"
    End With
    With .Controls.Add(Type:=msoControlPopup)
     .Caption = "&Fixture Unit"
'.OnAction = "ToBeDone"
    End With
    With .Controls.Add(Type:=msoControlButton)
     .BeginGroup = True
     .Caption = "FU & Flow Conversion"
     '.OnAction = "ShowfmFUQconvert"
    End With
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .Caption = "&Water Heating"
    With .Controls.Add(Type:=msoControlPopup)
     .Caption = "&Hot Water Demand"
'.OnAction = "ToBeDone"
    End With
    With .Controls.Add(Type:=msoControlButton)
     .Caption = "Water Heater Selection"
'.OnAction = "ToBeDone"
    End With
    With .Controls.Add(Type:=msoControlButton)
     .Caption = "Hot Water Circulation"
     .OnAction = "HWRecirculation"
    End With
    With .Controls.Add(Type:=msoControlButton)
     .Caption = "Expansion Tank"
'.OnAction = "ToBeDone"
    End With
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .BeginGroup = True
    .Caption = "&Drainage"
'.OnAction = "ToBeDone"
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .BeginGroup = True
    .Caption = "Gases"
'.OnAction = "ToBeDone"
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .BeginGroup = True
    .Caption = "Hydraulic Calc"
'.OnAction = "ToBeDone"
   End With
   With .Controls.Add(Type:=msoControlPopup)
    .BeginGroup = True
    .Caption = "Tools"
    With .Controls.Add(Type:=msoControlButton)
     .Caption = "Calculator"
'.OnAction = "ToBeDone"
    End With
    With .Controls.Add(Type:=msoControlButton)
     .Caption = "... ..."
'.OnAction = "ToBeDone"
    End With
   End With
   With .Controls.Add(Type:=msoControlButton)
    .Caption = "Help"
'.OnAction = "ToBeDone"
   End With
  End With
End Sub
```