Dustin Skinner

MBA 614

Table of Contents

Executive Summary

The Problem

It was about 2 years ago that I created a budgeting spreadsheet within Excel. Here, I tracked the many costs that both my wife and I had, expenses from education to entertainment. This helped to achieve a growth in savings, even amidst a tight budget. The ability to track the money that came in and out of our house was something interesting to me. After all, I am studying accounting, and this is a simple application of such a study. The problem arose when I wanted a better spreadsheet, but was not completely able to design one with the knowledge that I had. There were budgeting tools out there, but I didn't want to pay $60 to use them. There had to be some other way...

The Investigation

One day I was talking to a fellow classmate of mine about budgeting. He mentioned a budgeting software that he used called "You need a budget" or "YNAB." He said that while it cost $60 to purchase, it did allow a 1 week free installation to test-drive the software. I downloaded YNAB and quickly found ways that it was much better than the budgeting tool I was currently using. It had budgets for many different areas, but consolidated them so as to not overwhelm the user. It carried unused budgeted balances onto the next month, something I had not built into my budgeting spreadsheet. It was overall a very useful tool. Again, I bring up the cost. I, nor my wife, did not want to pay the $60 to purchase a budgeting software. It was the following week that I entered MBA 614 and we were to create something within Excel, using VBA, for a business use. I thought this would be the perfect time to create what I desired.

The Solution

My product is called iSYB, short for "I Stole Your Budget." With the week that I had YNAB, I learned the in's and out's and designed my own product within Excel. It has two spreadsheets, (1) Budget Detail, and (2) Charts. Both are used to help the person budgeting to see the details of his/her income and expenses. It allows both detailed and summary views of the data. Ultimately, iSYB is a budgeting solution for all those that can't fork out big money to help manage their cash flow.

iSYB Explanation

iSYB is best described in two parts. They are as follows:

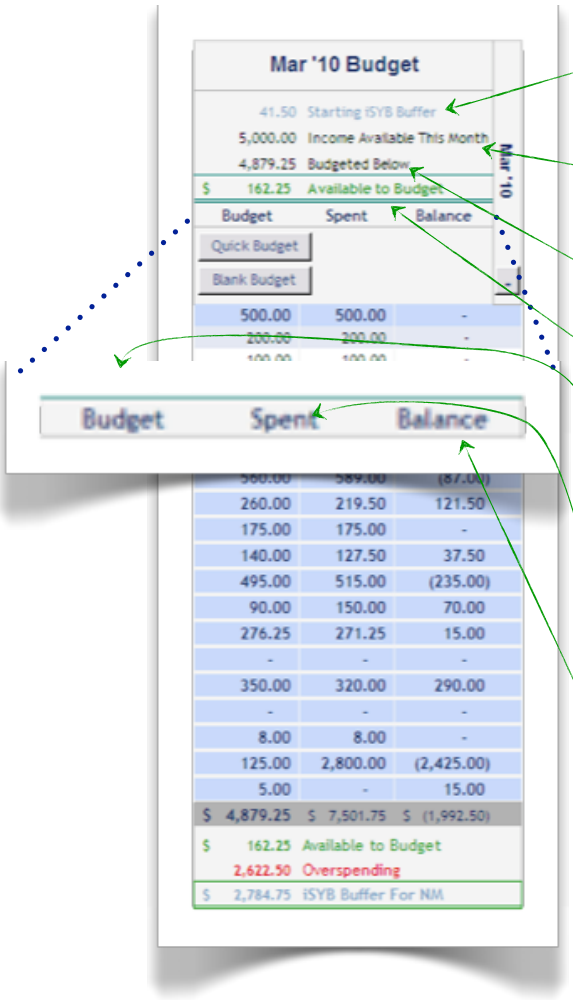
1. General Formatting/Usefulness
2. Buttons/Macros Used in Formatting

General Formatting/Usefulness

"Budget Detail" Worksheet

On the left is the main toolbar. Just below the logo there are hyperlinks to the other page in the spreadsheet ("Charts"). Below the toolbar are all the categories. Each categories has been included to cover nearly all necessary expenses. Each category has sub-categories that are a bit more specific. It is a frozen window so it is always visible no matter what month you are looking at.

To the right of the main toolbar are the various months. These months contain all the necessary data. Each month is grouped together and can be minimized or maximized in order to make viewing months easier. To explain the data and its use, below is a picture, along with the various parts of the budget explained:



- *Starting iSYB Buffer* - This is basically the buffer or savings that you have built up to that point. This can be thought of as what is in your checking account at the start of the month.
- *Income Available This Month* - This, just as the name explain, is the income that is earned in that particular month. This increases the amount that you are able to budget below.
- *Budgeted Below* - This is a total amount of everything that is budgeted in the categories below.
- *Available to Budget* - This is the amount of money that is available to budget without going below $0. If this amount does go below $0, the caption that says "Available to Budget" will change to say "Warning: Over-budget."
- *Budget* - This column includes all the budgeted amounts for the various categories and sub-categories. An amount is listed for a category, which is the total of all the sub-categories. The total of all budgeted amounts is what leads to the "Budgeted Below" figure listed above.
- *Spent* - This column is filled by the user and represent the actual costs that are incurred. Just as is understood by those using a budget, the user strives to maintain this amount below the budgeted amount listed (Unless it is an account with a balance built up).
- *Balance* - This is amount contains a couple values. First, it represents the difference between what is budgeted for a particular sub-category, and what has been spent in said sub-category. This amount is added to any remaining balance listed from the months prior. This amount will build for particular items such as budgeted amounts for doctor visits. While it is not fully expected to see the doctor every month, this balance will build in months that it is not used, so as to be ready for the month that it will be required.

"Charts" Worksheet

On the left, again we the see the main toolbar. This include the logo, along with the link back to the main worksheet, "Budget Detail."

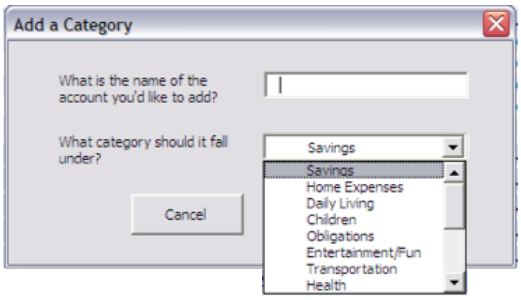
The center of the worksheet include the main information for the charts. The main part includes two buttons, along with blank area (this is where the chart will show up once the chart information is selected).

Buttons/Macros Used in Formatting

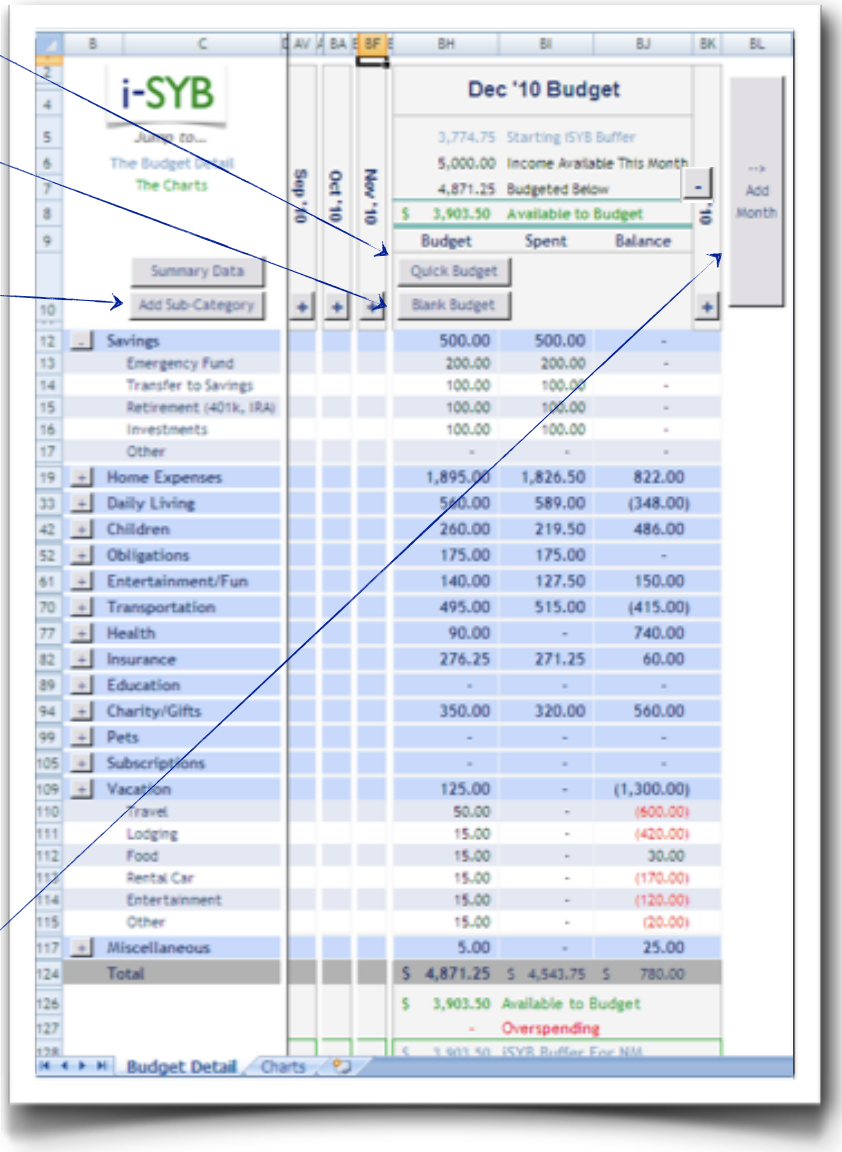
"Budget Detail" Worksheet

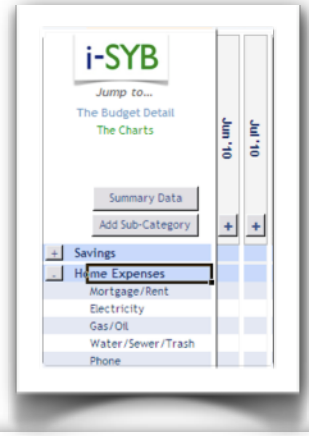
There are a couple useful buttons that help in the automation of the spreadsheet. They include: *Quick Budget*, *Blank Budget*, *Add Sub-Category*, *Add Month, Minimize/Maximize Feature,* and *Summary Data*.

1. Quick Budget is used to quickly fill in a new months blank budget with the same amounts of last month. This helps to quickly create a new month that is based on last month's figures.
2. *Blank Budget* is used to blank out a month's budget. This is used when you don't want to use the same information, but would rather see a blank sheet before creating your own budgeted figures.
3. *Add Sub-Category* is used to do just that, add a sub-category (A sub-category is basically an expense item). For example, under the "Savings" Category you might have a more specific sub-category called "Baby Fund." Once the sub-category button is clicked, it brings up a form (as shown below). In this form, you can type in a name for the new Sub-Category, as well as designate the Category under which the new Sub-Category will fall. All amounts are the new Sub-Category are automatically listed as $0.00.

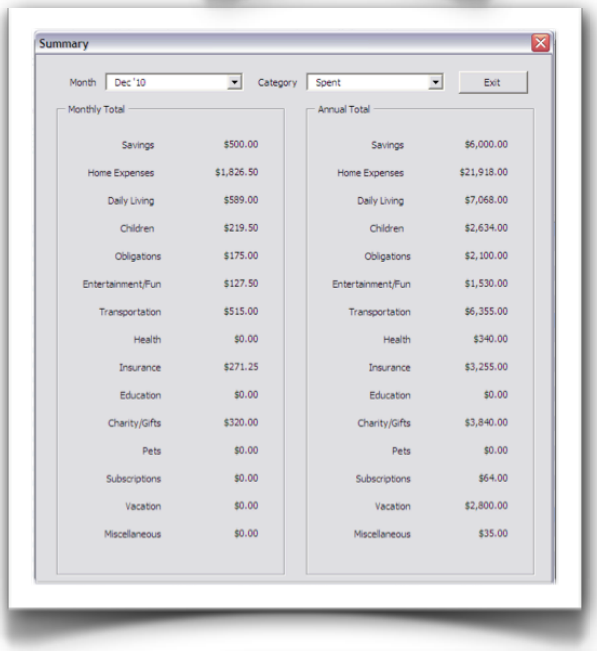


4. *Add Month* is used to add another month to your budget. As the budget continues to grow, additional months are required. This button automates the process. Once pressed, another group of columns (a new month) is automatically created. It code detects the prior month and renames the new month with the appropriately.





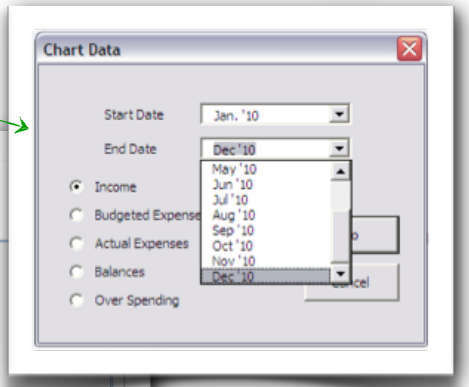
5. *Minimize/Maximize* is used to expand and contract various columns and rows. For example, the picture shows the month of December. By clicking on the button shown (The "-" symbol), it would minimize the month so December '10 would be minimized just as "Nov '10." By clicking on the same button, the month will be maximized again to show the month how it was originally depicted. Similarly, each category on the main toolbar can be minimized/ maximized to hide/show various sub-categories within each category. These options are provided to make it easier to organize and summarize the information.

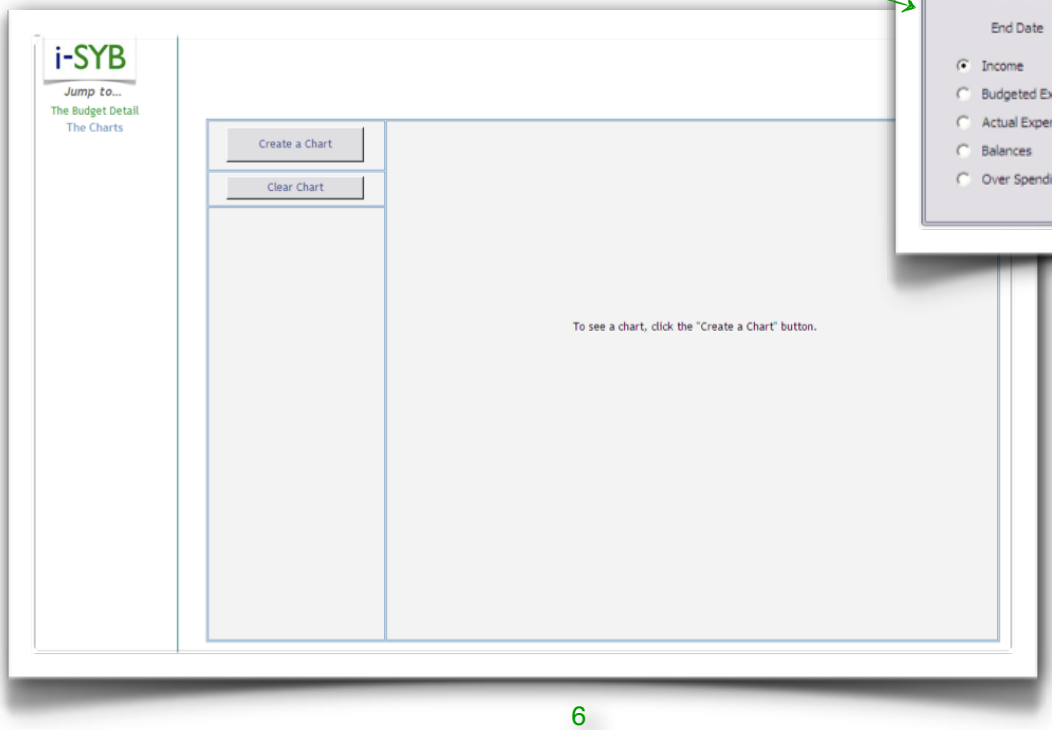


6. Summary Detail is macro that brings up the Summary user form. Once pressed, it opens a form that shows all the data for the given month and category (Budgeted, Spent, and Balance). The user form is shown on the right. The useful thing in this user form is that it shows the monthly data, as well as annual. For example, if the user wishes to see his/ her expenses for the month of December, the user will select "Dec. '10" from the Month drop-down combobox, as well as the "Spent" category from the Category drop-down combobox. The data will automatically update and show the monthly spent data for each category for the month of December, 2010, as well as the annual total up until that month. This is extremely useful if the user wants to easily see how much they have spent throughout the year on something like "Daily Living" expenses. If the user wants to close this user form, he/she simply presses the "Exit" button, or the "X" at the top right part of the window.

"Charts" Worksheet

Below is a picture that shows the "Charts" worksheet. As was mentioned before, there are two buttons, accompanied with a large area with a caption that say "To see a chart, click the "Create a Chart" button." Upon clicking the "Create a Chart" a user form appears like this.

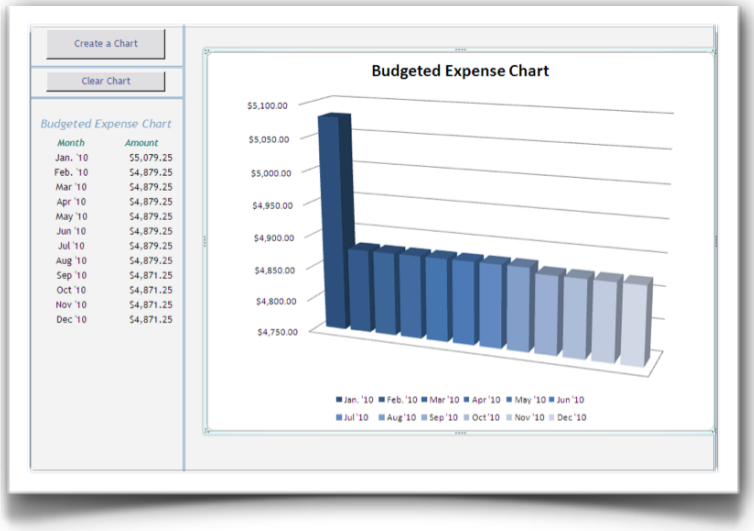




The user form automatically lists “Jan. ’10” (The first month in the budget) as the start date, and “Feb. ’10” as the end date (the second month in the budget). The dates selected represent the beginning and ending dates that will be used when the chart is created. After selecting the appropriate dates, a category will be selected. Choices include: *Income*, *Budgeted Expenses*, *Actual Expenses*, *Balances*, and *Over Spending* (These particular categories are as explained before in the “General Formatting/Usefulness” section on page 4). Once this category is selected, the user either presses enter, or selects “OK.” Upon clicking the “OK” button, the user will see a chart appear on the screen, as is shown by the picture to the right (This particular example shows the Budgeted Expenses from January 2010 through December 2010).

As is depicted, the chart shows the information in the chart, as well as two columns on the left part of the screen with the actual data that the chart is depicting.

If the user wants to create another chart, he/she simply presses the “Create a Chart” button once again and they will go through the same process that was just explained. If the user wishes to clear the chart, he/she will press the “Clear Chart” button. This will delete the chart from off the screen, as well as clear the data that is shown on the left columns.



Explanation of VBA Code

To explain the VBA code, I will list out the various buttons, as well as the various sub procedures that are used in their operation. First will be all those from the "Budget Detail" worksheet, followed by those from the "Charts" worksheet. I will also provide actual screenshots of portions of the sub procedures to aid in understanding (Some are too extensive to appear as one photo).

"Budget Detail" Worksheet

1. Quick Budget

The Quick Budget is a button that automatically updates the current month's budget with the numbers of last month. To do this, I first found a way to reference a range relative to the button that was pressed. This is done with the "TopLeftCell" action. As this is done, a range is set 3 columns down from the button address. This is given the same value from 5 cells to the left (last month's budgeted value). This references row 13, the first sub-category in the budget. From here, there is 2 Do Loops. The first runs until 2 rows down from the current range is blank. The next Do Loop sets the current range to the same cell, one row down. The value is then set as the same row, 5 columns to the left (which is last month's budgeted value). This Do Loop lies inside the first do loop, and will run until the row just below the current row is blank. This allows every cell to update until the very end, which has the following 2 rows with blank values. This will ultimately lead end the Do Loops and will end the sub. Screen updating is turned off throughout the entire sub procedure.

```
Sub QuickBudget()

    Dim s As String, btn As Button, r As Range
    Dim R1 As Range

    Application.ScreenUpdating = False

    s = Application.Caller
    Set btn = ActiveSheet.Buttons(s)
    Set r = btn.TopLeftCell

    Set R1 = r.Offset(3, 0)
    R1 = R1.Offset(0, -5)

    Do Until R1.Offset(2, 0) = ""

        Do Until R1.Offset(1, 0) = ""
            Set R1 = R1.Offset(1, 0)
            R1 = R1.Offset(0, -5)
        Loop

        Set R1 = R1.Offset(2, 0)

    Loop

    Application.ScreenUpdating = True

End Sub
```

2. Blank Budget

The Blank Budget is a button that automatically blanks out the current month's budget. This runs a very similar sub procedure as "Sub QuickBudget()" only instead of setting the values equal to the values of those from last month, they will be set to zero.

3. Add Sub-Category

Clicking on the button labeled "Add Sub-Category" brings up a user form with a TextBox and a ComboBox. The Textbox is used to label the new Sub-Category. The ComboBox is a list of all the current categories. The particular category selected will be the home of the new sub-category.

Once the user clicks "OK," the user form will run the AddRow2() sub procedure. There are two parts to this sub procedure: (1) Inserting a new row with the proper caption, and (2) filling the cells with either a "0" or a formula.

The first part basically matches the given input by the user from the ComboBox with the

```
Sub AddMonth()

    Dim s As String, btn As Button, r As Range
    Dim R1 As Range
    Dim Month As String

    Application.ScreenUpdating = False

    s = Application.Caller
    Set btn = ActiveSheet.Buttons(s)
    Set r = btn.TopLeftCell

    Set R1 = r.Offset(0, -5)
    R1.Columns("A:E").EntireColumn.Copy

    r.Offset(0, 0).Columns("A:A").EntireColumn.Select

    Selection.Insert Shift:=xlToRight

    ActiveCell.Offset(1, 1).Range("A1:C1").Select

    Month = ActiveCell.Value

    'This very long if statement is to decide what month name to put into "addmonth"

    If Left(Month, 3) = "Dec" Then

        Month = "Jan" & " '" & Mid(Month, 6, 2) + 1 & Right(Month, 7)
        ActiveCell.Value = Month
        ActiveCell.Offset(0, 1).Value = Left(Month, 7)

    ElseIf Left(Month, 3) = "Jan" Then

        Month = "Feb" & " " & Right(Month, 10)
        ActiveCell.Value = Month
        ActiveCell.Offset(0, 1).Value = Left(Month, 7)
```

```
    Set R1 = ActiveCell.Offset(0, 4)

    R1.Select

    Do Until R1.Offset(-1, 2) = ""

    Do Until R1.Offset(-1, 0) = ""

        If R1.Offset(-1, 0).HasFormula Then

            R1 = R1.Offset(-1, 0).FormulaR1C1

        ElseIf R1.Offset(-1, 0) = "" Then

            R1 = R1.Offset(-1, 0)

        ElseIf Not R1.Offset(-1, 0) = "" And Not R1.Offset.HasFormula Then

            R1 = 0

        End If


        Set R1 = R1.Offset(0, 1)

    Loop

    Set R1 = R1.Offset(0, 1)

    Loop

    ActiveCell.Offset(0, -2).Select

    CategoryForm.Hide

    End

End Sub
```

given category from Column B. Once there is a match found, the code will copy the second to last row and insert another row as the last sub-category (this assures that the formatting continues in "Blue-White" succession). The appropriate caption is then placed into the new sub-category row.

Next, the code will fill in the new values for the new sub-category. It is not as simple as putting in 0's for every. Instead, it will recognize if the row above has a number or a formula. If it is a number, it will insert a "0" to leave the budget/ spent value as blank. If there is a formula, it will copy the formula down (this assures that the balance formula works for new sub-categories). This particular part of the code is shown on the left.

4. Add Month

Add Month has a similar feature to the Quick/Blank Budget buttons that recognizes the range where the button "Add Month" button is located. This helps to automate the process of adding columns just after the last month. The first part of the sub procedure is simple as it copies the previous 5 columns (an entire month segment) and pastes/inserts this into the columns just before the column where the "Add Month" button is located.

The next portion of this sub procedure is very long and detailed, but fairly simple to understand. The cell that describes the month is selected as the active cell. The string "Month" is given this value. From here, there are a series of If...Then...ElseIf statements built in to recognize what month and year is in that string. Using the "Left" feature in VBA, the month is recognized. Recognizing this month, the If statements then give the next month's name as the new value. If the previous month was listed as December, the "Mid" feature is used to add 1 to the year (2009 + 1 = 2010).

After this is complete, it runs basically the same code as the "Blank Budget" sub procedure, which zeroes out the entire budgeting column.

5. Minimize/Maximize

This particular sub procedure draws upon one of two sub-procedures: HideDetail (HideMonth) or ShowDetail (ShowMonth). The sub procedure uses an If Statement to determine if their is hidden data or not. If it detects hidden data, it will perform the ShowDetail (ShowMonth) sub procedure, which will unhide the rows for that particular procedure (ShowMonth will do the same, but for columns for that particular month). The works just the opposite if it detects that rows/columns are not hidden by performing the HideDetail (HideMonth) sub procedure. Along with this, the caption of the button will change according to which sub procedure is called.

6. Summary Detail

This sub procedure was a hand-full. Two arrays are filled (one 3-dimensional array, another 4-dimensional array). One is filled for the Monthly Data, another is filled for the Annual Data. This data is filled upon selection of month and category when the user form is opened.

The 3-dimensional array includes (Month, Category Type, and Amount). This is the MonthlyArray.

The 4-dimensional is similar, only the month is divided out into a year in one dimension, and a month in another. This array is made up of the MonthlyArray, and it is filled using the code shown below.

```
Do Until a = 11
        x = 1
        Do Until x = 13
            b = 1
            If MonthNameArray(x2) = "" Then
                Exit Do
            End If
            Do Until b = 16

                If (x2 - 1) Mod 12 = 0 Then

                    YearlyArray(a, y, x, b) = MonthlyArray(x2, y, b)
                    YearlyArray(a, y + 1, x, b) = MonthlyArray(x2, y + 1, b)
                    YearlyArray(a, y + 2, x, b) = MonthlyArray(x2, y + 2, b)

                Else

                    YearlyArray(a, y, x, b) = YearlyArray(a, y, x - 1, b) + MonthlyArray(x2, y, b)
                    YearlyArray(a, y + 1, x, b) = YearlyArray(a, y + 1, x - 1, b) + MonthlyArray(x2, y + 1, b)
                    YearlyArray(a, y + 2, x, b) = MonthlyArray(x2, y + 2, b)

                End If

                b = b + 1
            Loop
            x = x + 1
            x2 = x2 + 1
        Loop
    If MonthNameArray(x2) = "" Then
        Exit Do
    End If
    a = a + 1
Loop
```

Basically, the array will go through each category and will up every amount required and will then move on to the next month in the particular year. Once it hits the 12th month, it will move on to the next year and re-perform the previous action. These loops will continue until the month or year is beyond what is listed in the "Budget Detail" worksheet.

These arrays are then loaded onto the "Summary Detail" user form as the text captions are replaced with the values from the arrays (whether it is monthly or annual).

"Charts" Worksheet

1. Create a Chart

The "Create a Chart" feature does not store the actual data in any particular worksheet or cell, so the code is rather extensive.

The user form is first opened with the value previously mentioned in the formatting section (Pg. 6). When the user selects the values he/she wants, these values are loaded into various variables. The AddChart() sub procedure is then called.

First, each of the ComboBox selections and button values are passed to the AddChart() sub procedure.

Next, a new chart is created. This chart will be manipulated later according to the data that was selected on the user form.

Next, a Do Loop runs that collects the various months and saves them in a array. These values are then matched with the months that were selected for the start and end date for the chart. The program then verifies that the end month is equal to or after the beginning month. If not, it presents an error box that explains the problem, and then closes the user form.

The sub procedure then runs a series of If statements that recognize what category was selected, and then names the charts, as well as the detail on the “Charts” worksheet accordingly. Also, whatever category that was selected, it will select that particular range (if the “Budgeted Expenses” category is selected, it will load the value from Range “I-13” into the DataCell range. This particular segment is shown with the screenshot to the right.

```
If Income = True Then
    Set DataCell = Sheets("Budget Detail").Range("E6")
    ActiveChart.ChartTitle.Text = "Income Chart"
    Sheets("Charts").Range("I13") = "Income Chart"
ElseIf BExpenses = True Then
    Set DataCell = Sheets("Budget Detail").Range("E124")
    ActiveChart.ChartTitle.Text = "Budgeted Expense Chart"
    Sheets("Charts").Range("I13") = "Budgeted Expense Chart"
ElseIf AExpenses = True Then
    Set DataCell = Sheets("Budget Detail").Range("F124")
    ActiveChart.ChartTitle.Text = "Actual Expense Chart"
    Sheets("Charts").Range("I13") = "Actual Expense Chart"
ElseIf Balance = True Then
    Set DataCell = Sheets("Budget Detail").Range("G124")
    ActiveChart.ChartTitle.Text = "Balances Chart"
    Sheets("Charts").Range("I13") = "Balances Chart"
ElseIf OSpending = True Then
    Set DataCell = Sheets("Budget Detail").Range("E128")
    ActiveChart.ChartTitle.Text = "Over-Spending Chart"
    Sheets("Charts").Range("I13") = "Over-Spending Chart"
End If
```

```
    Do Until MonthName(x) = MonthName(mns)
        Set DataCell = DataCell.Offset(0, 5)
        Set DataMonth = DataMonth.Offset(0, 5)
        x = x + 1
    Loop

    x = 1
    Set DataInput = Sheets("Charts").Range("I15")
    Sheets("Charts").Range("I14").Value = "Month"
    Sheets("Charts").Range("J14").Value = "Amount"

    Do Until MonthName(mns) = MonthName(mne + 1)

        ActiveChart.SeriesCollection.NewSeries
        ActiveChart.SeriesCollection(x).Name = DataMonth
        ActiveChart.SeriesCollection(x).Values = DataCell

        DataInput = DataMonth.Value
        DataInput.Offset(0, 1) = DataCell.Value

        Set DataMonth = DataMonth.Offset(0, 5)
        Set DataCell = DataCell.Offset(0, 5)
        Set DataInput = DataInput.Offset(1, 0)

        mns = mns + 1
        x = x + 1
    Loop

    Dim RngToCover As Range
    Dim ChtOb As ChartObject

    Set RngToCover = ActiveSheet.Range("M9:T34")
    Set ChtOb = ActiveChart.Parent
    ChtOb.Height = RngToCover.Height ' resize
    ChtOb.Width = RngToCover.Width   ' resize
    ChtOb.Top = RngToCover.Top       ' reposition
    ChtOb.Left = RngToCover.Left     ' reposition

End Sub
```

After this, it recognizes what the start month is, and accordingly moves the DataCell range to the appropriate category data (This part is shown in the screenshot to the left). This means that it will select the first month and load the appropriate data from that month into the DataCell. If the start month is not equal to that particular month, it will select the next month and load its respective data in DataCell. It will continue this until the start date is equal to the month selected. From here, the chart begins to be populated until the final month is recognized. Last of all, the program resizes and refits the chart to fit inside the appropriate area of the worksheet. The user form is then closed and the graph is shown according the data selected.

2. Clear Chart

The “Clear Chart” button runs a sub procedure named ClearCharts(). This is a simple procedure that runs through al the chart objects in the “Charts” worksheet and deletes them using a For Loop. This sub procedure is depicted in the screenshot to the right.

```
Sub ClearCharts()

    Application.ScreenUpdating = False

    Dim ws As Worksheet
    Dim ChartO As ChartObject

    For Each ws In ThisWorkbook.Worksheets

        For Each ChartO In ws.ChartObjects

            ChartO.Delete

        Next

    Next

    Sheets("Charts").Range("I12:J36").Select
    Selection.ClearContents
    Sheets("Charts").Range("I12").Select

    Application.ScreenUpdating = True

End Sub
```

Learning/Difficulties

Learning

Throughout the process of designing and writing the code for this project, I learned quite a lot that I did not previously know. Prior to this, I had a very limited understanding of arrays. After working out the Summary Detail form, I found that the best way to do it was with a multiple dimension array. I ended up having to use a 4-dimensional array that was then used in 4 nested Do Until loops. It was very complicated code that I previously would have never been able to do.

A lot of what I learned to do came from simple searches on google. If ever I was having trouble trying to look up a particular thing to do (such as working with arrays), I found that I was able to find answers by many various different websites on the internet.

Difficulties

I did run into a problem there and again. A few are listed below:

1. Minimize/Maximize caption is occasionally incorrect after running the Summary Detail. This is due to the fact that I had to maximize all of the categories in order to load the arrays for the Summary Detail. I was unable to change the various captions due to my inability to locate particular buttons throughout the spreadsheet.
2. Due to the Summary Detail having to maximize all the categories to load the data into the arrays, it left the “Budget Detail” spreadsheet differently then how it was left when the “Summary Detail” button was clicked. As a consequence, if some categories were minimized, all would become maximized after clicking on the Summary Detail.
3. I would work on my project in both Excel 2007 and Excel 2003. After creating some of my sub procedures in Excel 2007, I had trouble running some of them on Excel 2003 (most particularly the charts, which did not have the particular design I was calling for in the VBA code).

Overall, I feel that most of the trouble that I ran into I was able to solve. I learned a great deal in designing this budget. I’m grateful we had this project as I was able to save about $60 in the process.