MoneyTracker

The easy solution to track and save your money

Thom Archibald
4/13/2011

ISYS 540

EXECUTIVE SUMMARY

PROBLEM HISTORY

From a young age, I have always been taught the importance of saving money, and ultimately spending less than I earn. When I started a family of my own, the need to save money turned from a nicety to a necessity, with every dollar of income being so valuable to our getting by each month. As I approach the end of my "student life" at BYU, my wife and I are determined to continue with our spending habits, even though we will be making "real money" with my new full-time position.

In the past, we have managed our finances through a simple but customized spreadsheet. It works great, but requires a lot of data entry for any expenditure or income. The current process is that the receipts stack up in our wallets for about a month. Then, when we realize that nothing has been done to record our expenses for the last month, we pile them up by the computer until one of us enters them all into the spreadsheet and views the results. This process was screaming for automation, and this class created the perfect opportunity to better track and manage our finances. After all, the first step to saving money is to realize where your money is going.

CREATING A SOLUTION

My vision began with a desire to create a solution that I could hand off to family and friends and allow them to use the same program with virtually no training. I wanted it to be robust enough to handle new users in completely new environments without errors or unexpected activity. First, I wanted to create a way that users could dynamically manage their customized categories for expenses and income. Additionally, I realized the benefit of importing the data from our credit card and bank websites, making it much more likely that we would stay more up to date on tracking our expenses. With the help of Dr. Allen's agent class, I was able to import the data and present it to the user one transaction at a time for categorization and editing.

IMPLEMENTATION

The implementation of Money Tracker covers 5 main areas, each described below: User Interfaces, Adding Transactions, Accessing Online Data, Managing Categories, and Displaying Totals.

USER INTERFACES

The User Interfaces within Money Tracker are minimalistic and easy to use. The different worksheets within the workbook are hidden, so navigation throughout the program is accomplised with buttons. The main screen presents the users with four options: *New Transactions*, *View Transactions*, *View Expense Totals*, and *View Income Totals*. *New Transactions* brings up a simple user form that allows for new transactions to be entered into the program (see Figure 2), and handles importing of new transactions from online sources. *View Transactions* simply takes the user to the worksheet that stores all transaction information in two tables (see Figure 10). *View Expense Totals* and *View Income Totals* dynamically create pivot tables in the main screen to display summary information from all of the transactions (see Figures 8 and 9).

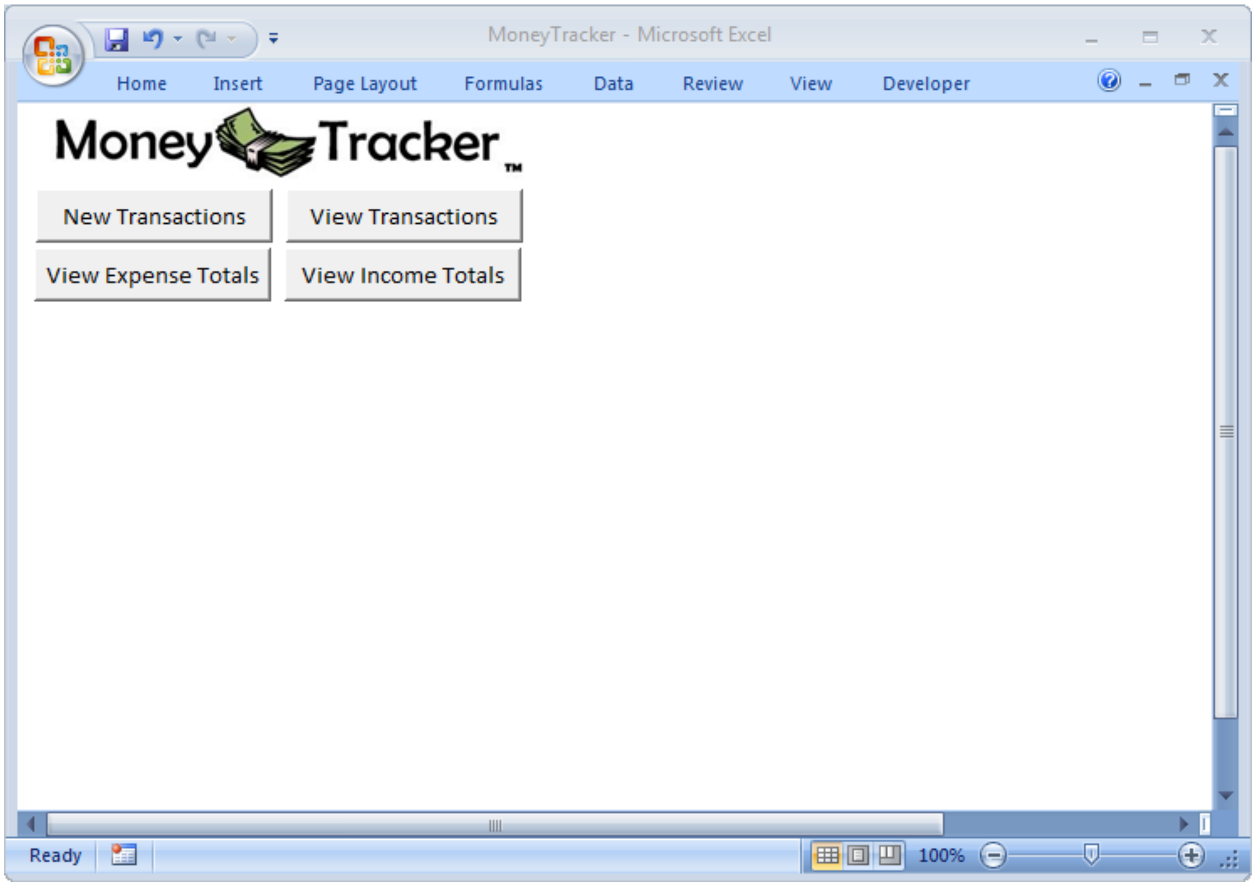


Figure 1: Main Screen

ADDING TRANSACTIONS

Adding a new transaction is a staple functionality for an expense-tracking program. From the home screen, the user selects *New Transactions* and views the *New Transaction* form (Figure 2). This form allows the entry of a new income (Figure 3) or expense (Figure 4). Based on which item the user selects, the corresponding fields are enabled or disabled. After entering all necessary data, the user can simply click *Next Transaction* to copy the transaction information to the spreadsheet for storage. The fields are then left blank for input of the next transaction. At anytime the form can be closed via *Cancel.*

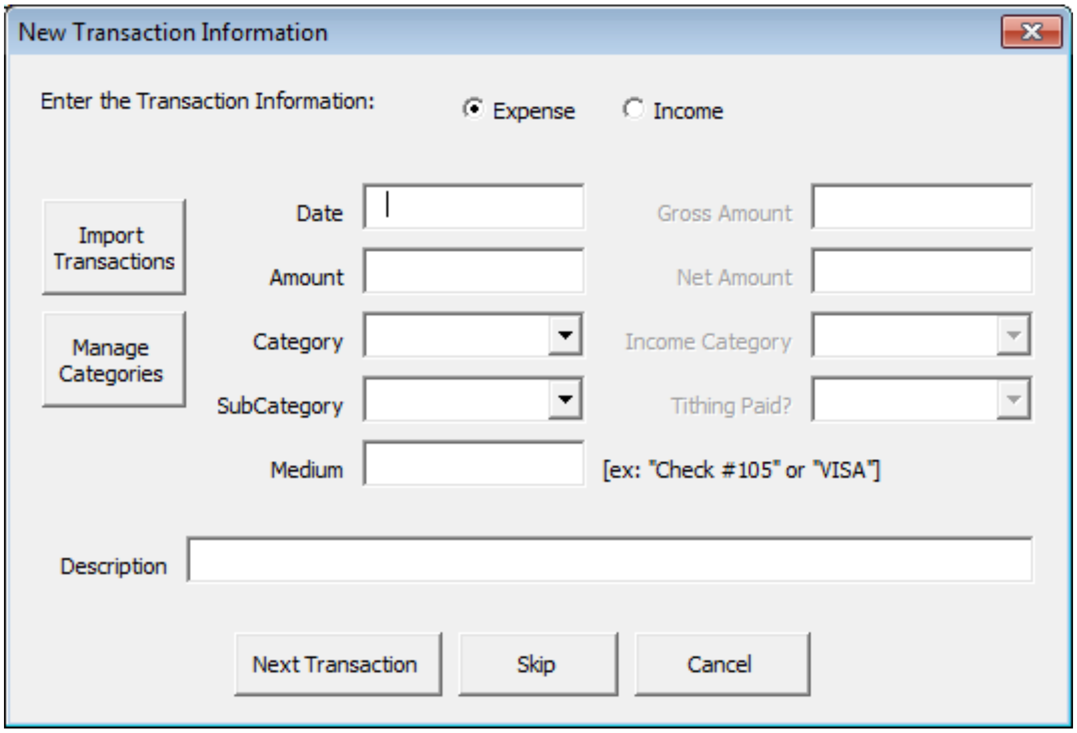


Figure 2: New Transaction Form - Expense

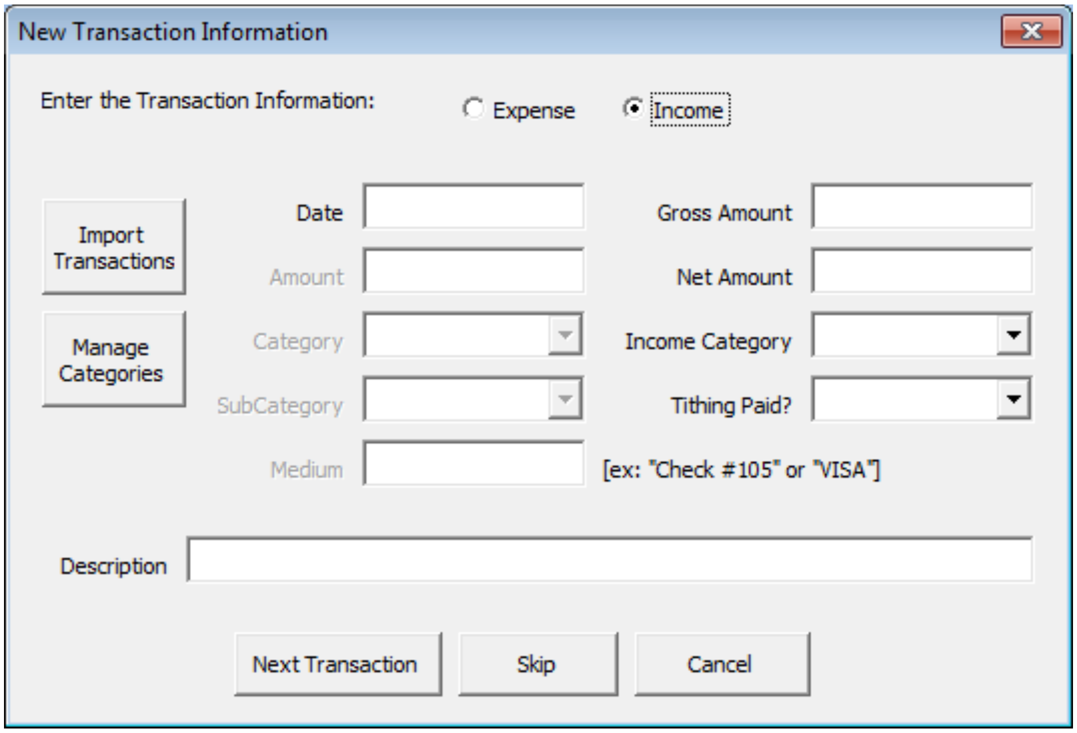
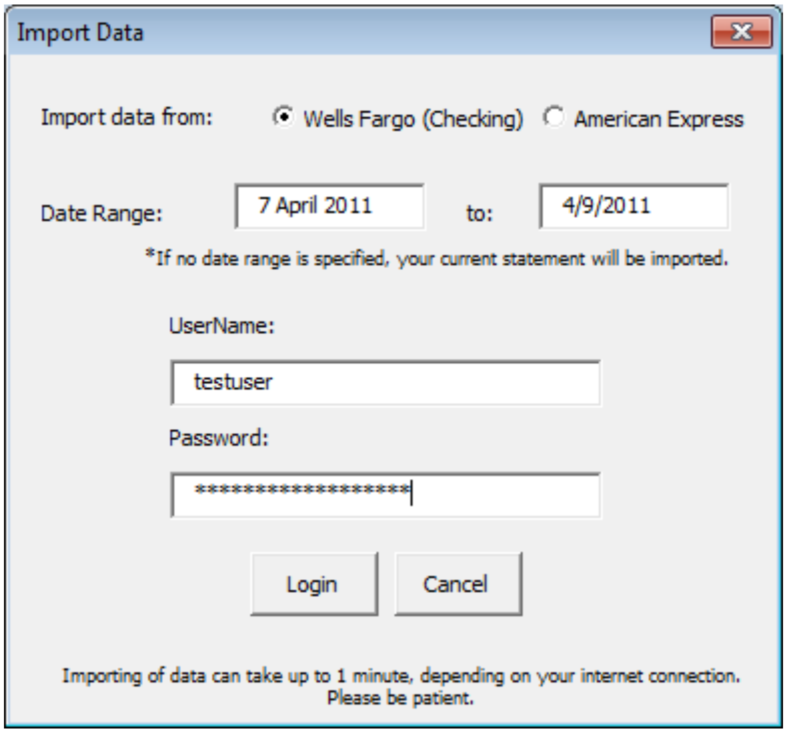


Figure 3: New Transaction Form - Income

ACCESSING ONLINE DATA

The next function of Money Tracker is to import transactions from two websites: wellsfargo.com, and americanexpress.com. This minimizes user entry, but still gives the user an individual transaction idea of where their money is going. From the *New Transactions* form, the user clicks on *Import Transactions* and is presented with the *Import Data* form (Figure 4). This form allows for import of data from either Wells Fargo or American Express by date, or by the current statement if the date fields are left blank.



Import data from: [✓ Wells Fargo (Checking)] American Express

Date Range: 7 April 2011 to: 4/9/2011

*If no date range is specified, your current statement will be imported.

UserName: testuser

Password: ******************

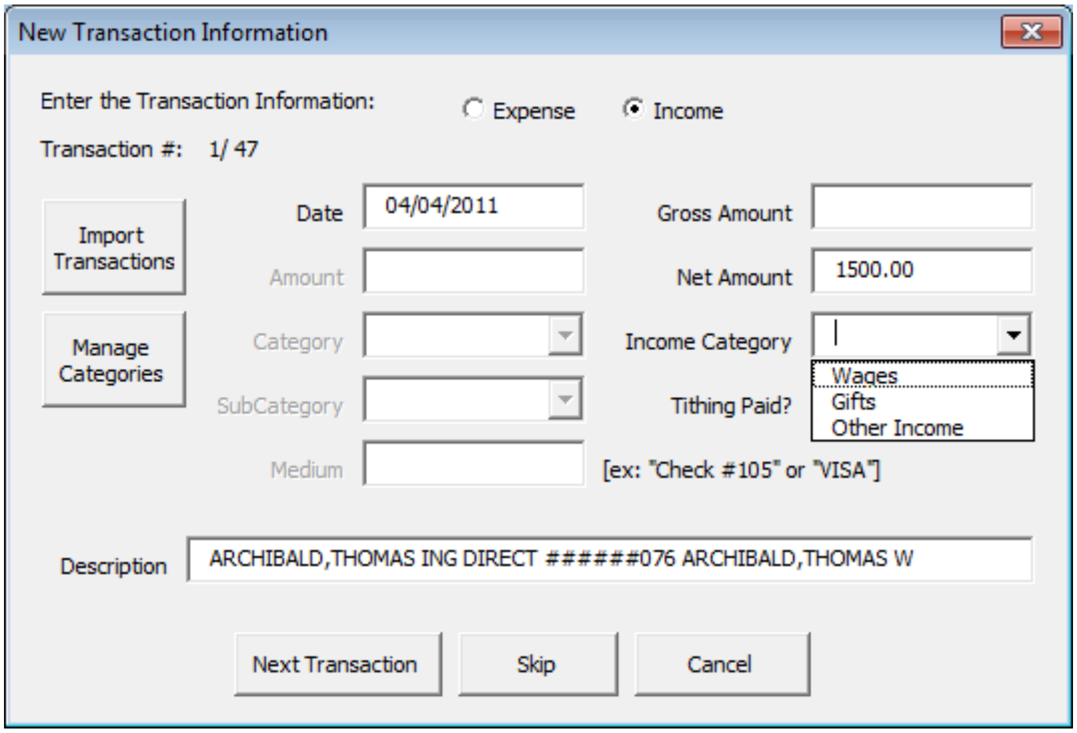
Importing of data can take up to 1 minute, depending on your internet connection. Please be patient.

Figure 4: Import Data Form

One great aspect of this form is that the date values can be entered in any Excel-acceptable date format, and they are then converted to the format that is acceptable on the right website. When the user clicks *Login*, the agent class is utilized to access the correct site and download a CSV file of the requested data. The file is stored temporarily in a “_data” folder that exists in the same path as the workbook.

Once this information is downloaded, it is read into an array and presented to the user one transaction at a time in the *New Transaction* form (Figure 5). I wanted to be able to have a balance of user input and automation so that I still am aware of each individual transaction. With this feature, I can classify any values that are downloaded in my own way, or skip them all together. For values where the account balance was increased in Wells Fargo, it defaults to an income where the user can choose from one of their categories, specify the Gross Amount, Tithing Paid, and alter the given description. For other values, it defaults to an expense and the user can specify all the transaction information for it as well. Because the values are being stored in a table within another spreadsheet (accessible from the main menu), users are then able to filter transactions by any of these fields, and on any level.

For transactions imported from Wells Fargo, it parses out the check number in a transaction and fills it in the *Medium* field. For all other values imported, it fills the default values of “Wells Fargo” or “AMEX” based on the source of the import. As with all other values in the form, these can be edited based on the user’s personal needs. Wells Fargo also attaches a double quote at the beginning and ending of each CSV record, because commas are included in some values (as evidenced in the *Description* field in Figure 5 below). The code also accounts for the presence of the minus sign preceding any expense (or debit) from the checking account by parsing it off. Because the values are categorized separately for incomes and expenses within the program, all values should be positive when entered. A portion of the code used to parse the Wells Fargo CSV file is included below in Figure 6.


New Transaction Information

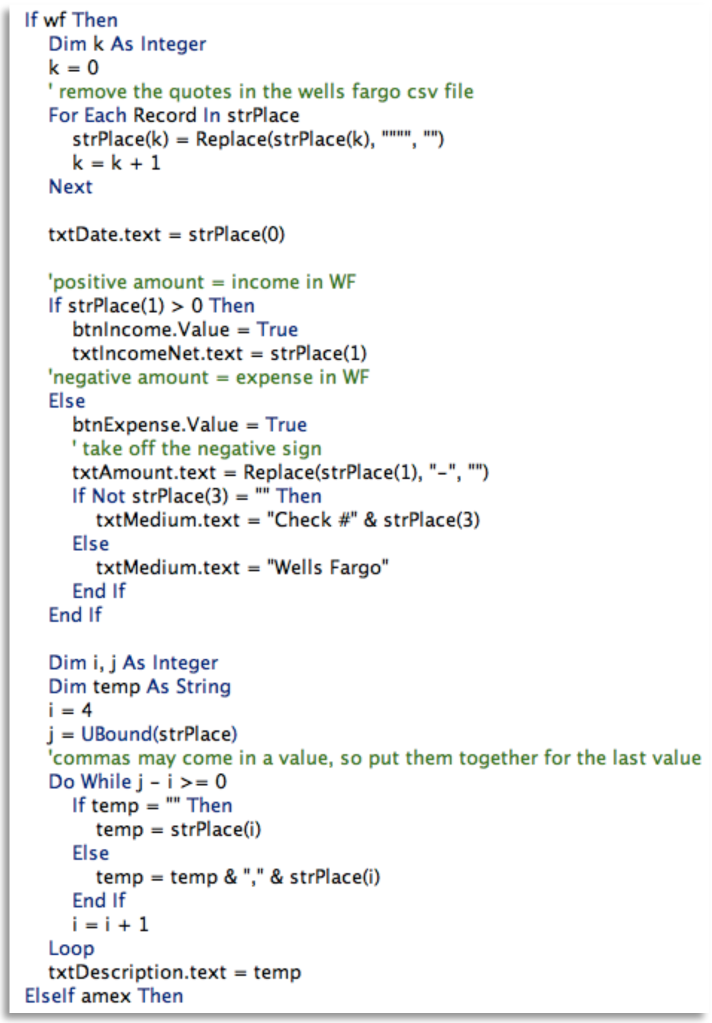
Enter the Transaction Information: Expense [✓ Income]

Transaction #: 1/ 47

Date	04/04/2011	Gross Amount	
Amount		Net Amount	1500.00
Category		Income Category	
SubCategory		Tithing Paid?	
Medium		[ex: "Check #105" or "VISA"]	

Description: ARCHIBALD,THOMAS ING DIRECT ######076 ARCHIBALD,THOMAS W

Figure 5: New Transaction - Importing Online Data



```
If wf Then
    Dim k As Integer
    k = 0
    ' remove the quotes in the wells fargo csv file
    For Each Record In strPlace
        strPlace(k) = Replace(strPlace(k), """", "")
        k = k + 1
    Next

    txtDate.text = strPlace(0)

    'positive amount = income in WF
    If strPlace(1) > 0 Then
        btnIncome.Value = True
        txtIncomeNet.text = strPlace(1)
    'negative amount = expense in WF
    Else
        btnExpense.Value = True
        ' take off the negative sign
        txtAmount.text = Replace(strPlace(1), "-", "")
        If Not strPlace(3) = "" Then
            txtMedium.text = "Check #" & strPlace(3)
        Else
            txtMedium.text = "Wells Fargo"
        End If
    End If

    Dim i, j As Integer
    Dim temp As String
    i = 4
    j = UBound(strPlace)
    'commas may come in a value, so put them together for the last value
    Do While j - i >= 0
        If temp = "" Then
            temp = strPlace(i)
        Else
            temp = temp & "," & strPlace(i)
        End If
        i = i + 1
    Loop
    txtDescription.text = temp
ElseIf amex Then
```

Figure 6: Wells Fargo Import Code

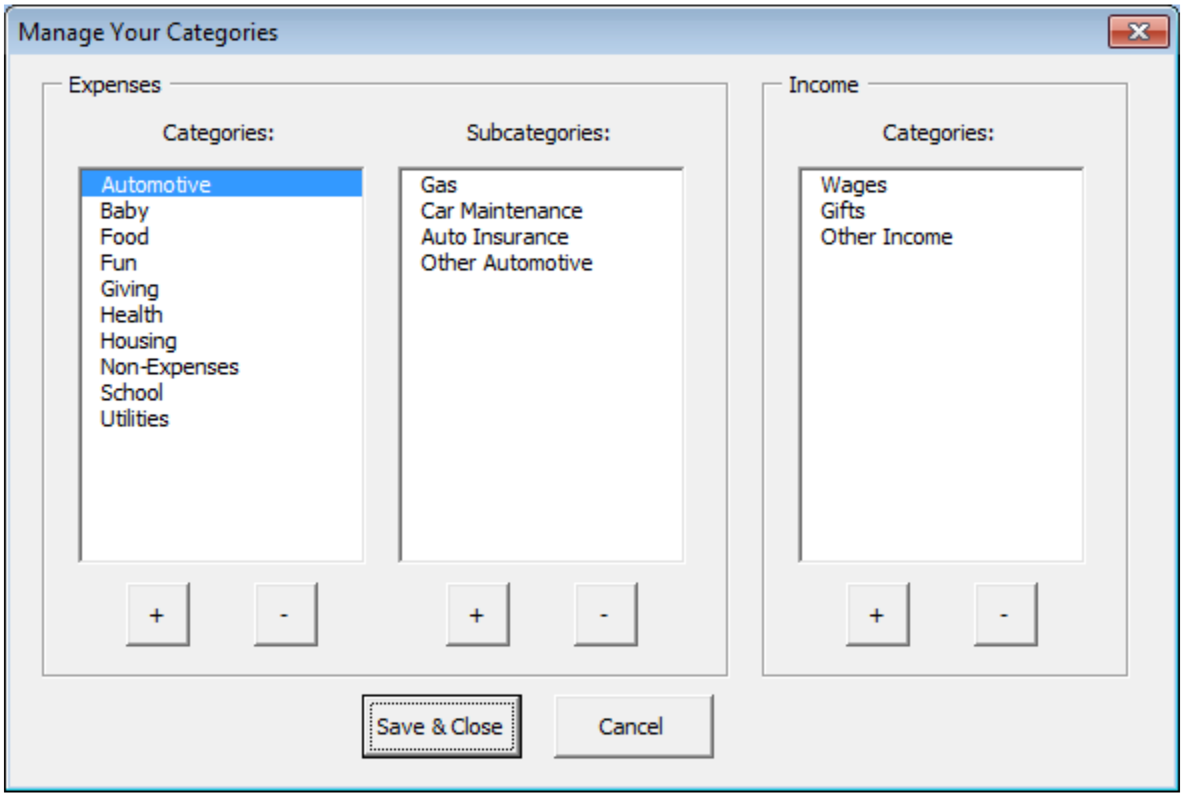
If a value appears that is not necessary to appear as a transaction, the user can press *Skip*, which displays the next record without saving the previous one. In this form, when transactions are imported, a transaction number is displayed in the top left corner to let the user know how many more transactions there are in the imported file.

MANAGING CATEGORIES

Categories and subcategories are dynamically managed through the *Manage Categories* button in the *New Transaction* form, displaying the *Manage Categories* form (Figure 7). The category and subcategory data are stored on a separate hidden worksheet, and updated immediately when the categories are added or removed. Each category can have many subcategories, so when the user selects a category the relating subcategories are display in the subcategory list. Items can be added and removed as desired, and when a category is removed, the user is warned that all relating subcategories will be removed as well. Income Categories can be managed here as well through simple adding and removing, via the "+"

and “-“ buttons below each list. The form will make sure that a category is selected before it can be removed, and correlates the proper category with added subcategories.

This data is used throughout the rest of the program for the user to classify their transactions. For example, when the *New Transaction* form is loaded, the category and subcategory combo boxes are populated based on the values entered into this form.

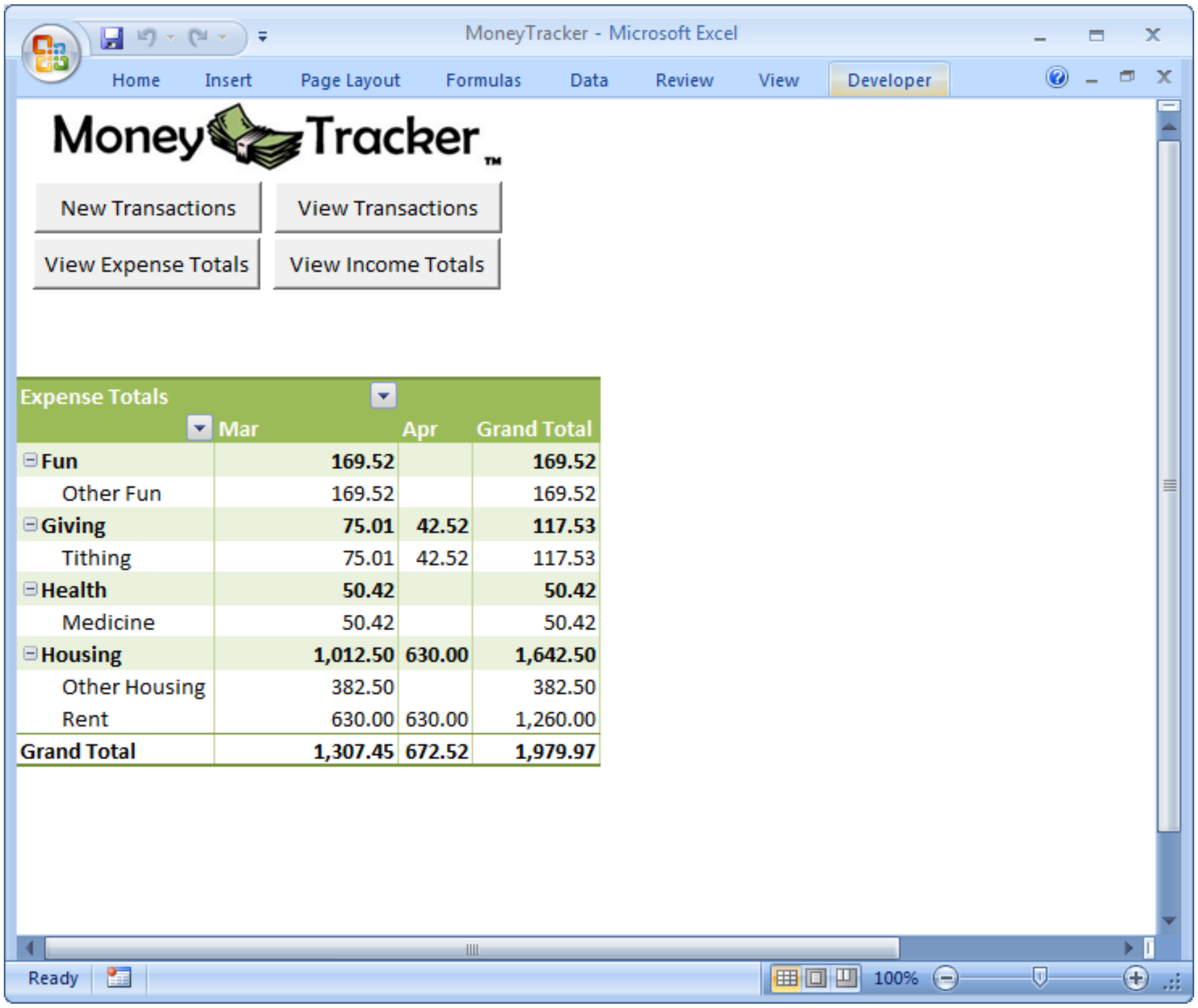


Expenses Categories	Expenses Subcategories	Income Categories
Automotive	Gas	Wages
Baby	Car Maintenance	Gifts
Food	Auto Insurance	Other Income
Fun	Other Automotive	
Giving		
Health		
Housing		
Non-Expenses		
School		
Utilities		

Figure 7: Manage Categories Form

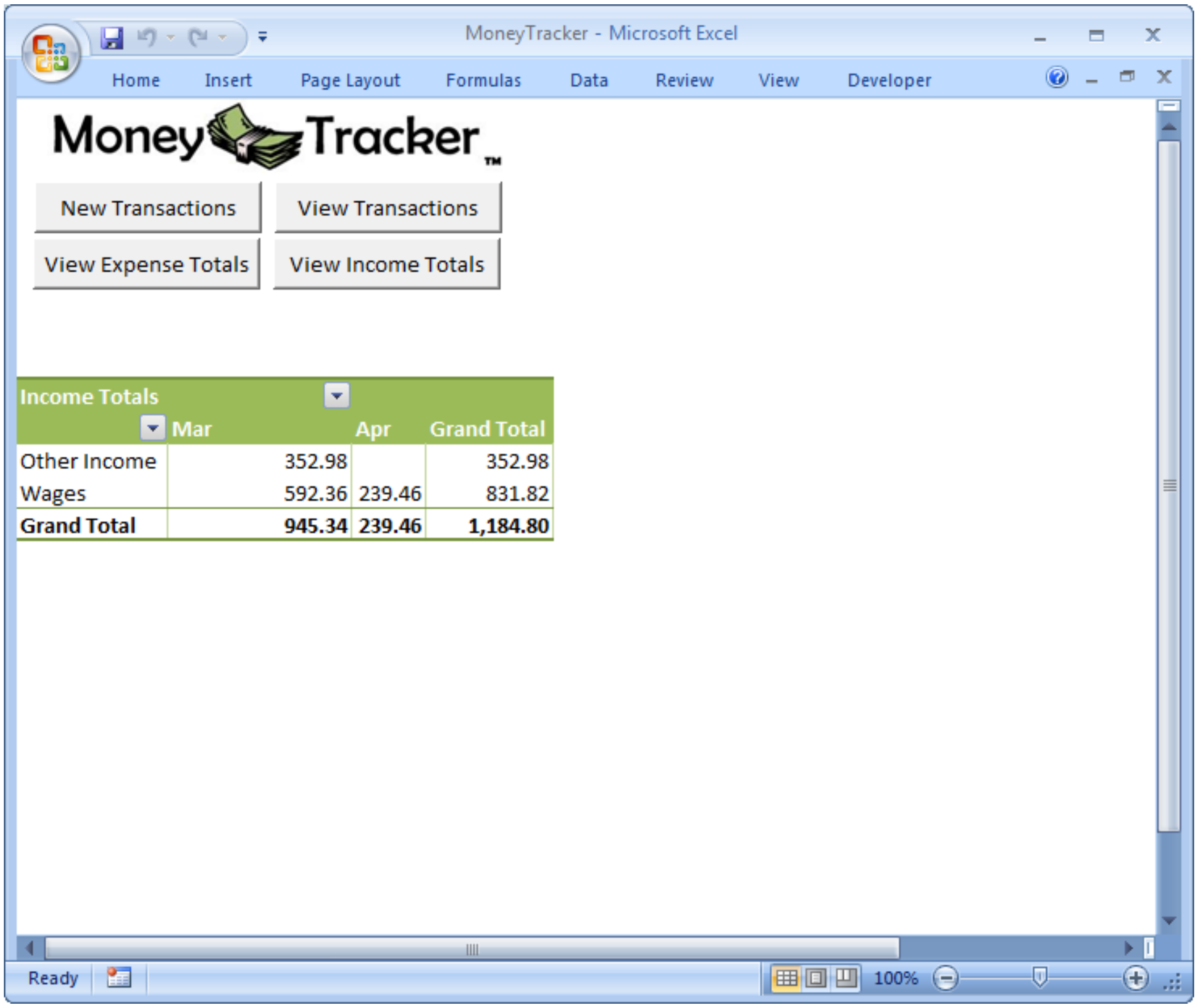
DISPLAYING TOTALS

A final important functionality of Money Tracker is the utilization of Excel’s Pivot Tables. From the main menu (Figure 1), the user can select *View Expense Totals* or *View Income Totals*, each of which creates a dynamic Pivot Table through VBA in the space below (see Figures 8 and 9). These pivot tables group by month along the top, and total the transaction amounts for each category and subcategory in the data fields. They also include totals for each category, month, and a grand total, each providing valuable “at-a-glance” data for the user.



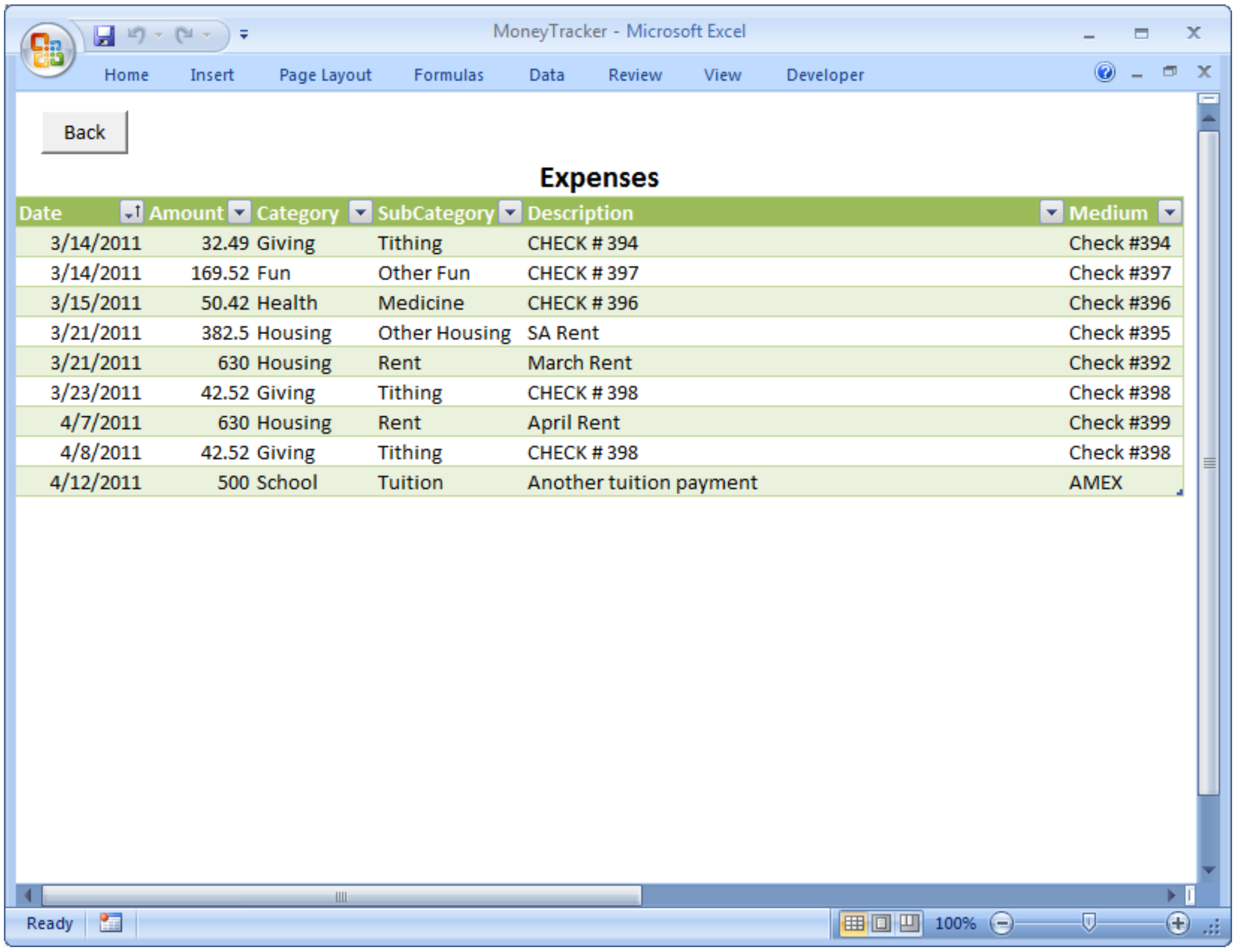
Expense Totals	Mar	Apr	Grand Total
Fun	**169.52**		**169.52**
Other Fun	169.52		169.52
Giving	**75.01**	**42.52**	**117.53**
Tithing	75.01	42.52	117.53
Health	**50.42**		**50.42**
Medicine	50.42		50.42
Housing	**1,012.50**	**630.00**	**1,642.50**
Other Housing	382.50		382.50
Rent	630.00	630.00	1,260.00
Grand Total	**1,307.45**	**672.52**	**1,979.97**

Figure 8: Expense Totals Pivot Table



Income Totals			
	Mar	Apr	Grand Total
Other Income	352.98		352.98
Wages	592.36	239.46	831.82
Grand Total	**945.34**	**239.46**	**1,184.80**

Figure 9: Income Totals Pivot Table



Expenses

Date	Amount	Category	SubCategory	Description	Medium
3/14/2011	32.49	Giving	Tithing	CHECK # 394	Check #394
3/14/2011	169.52	Fun	Other Fun	CHECK # 397	Check #397
3/15/2011	50.42	Health	Medicine	CHECK # 396	Check #396
3/21/2011	382.5	Housing	Other Housing	SA Rent	Check #395
3/21/2011	630	Housing	Rent	March Rent	Check #392
3/23/2011	42.52	Giving	Tithing	CHECK # 398	Check #398
4/7/2011	630	Housing	Rent	April Rent	Check #399
4/8/2011	42.52	Giving	Tithing	CHECK # 398	Check #398
4/12/2011	500	School	Tuition	Another tuition payment	AMEX

Figure 10: View Transactions

LEARNING DIFFICULTIES

The most difficult parts of this project had to do with accessing the online data through use of the agent class. Once I was able to retrieve the data successfully from the websites, the rest came more easily. Over time, I was able to solve all of the problems I faced, and accomplished my objectives for the project – and then some. I initially wanted to just create a way to dynamically manage categories and transactions through the simply interface and user forms, but I was pleased to be able to access online data and incorporate that into my project as well.

CONCLUSION

Money Tracker will be a valuable asset to my family as we seek to manage our finances into the coming months and years. I will continue to improve this project as needs for advanced functionalities arise, but I am very happy with how it turned out.