

Executive Summary

My project consists of two Excel VBA programs that open and operate an online project management tool called Mariner. BYU uses this system to keep track of their own projects, but the system falls short on speed and batch capabilities. Therefore, I decided to focus my project on speeding up the process through VBA and Excel.

Attribute Creator

Within Mariner, a user can create custom screens with the information of their choosing. However, to make a screen the designer must first make the UI objects and attributes within the database. For example, I want a check box that tells me if the project is on hold or active. I create a check box object and then assign an attribute to the database and the checkbox object. Now I can use that checkbox on a screen and it will already be set up to talk to the database.

One screen can contain as many attribute objects as needed, but to create a large number of attributes can be time consuming. This is due to two things:

1. The naming convention of the attributes is very specific. The database holds the names of over 400 and if the naming is unclear then we may run into problems in the future when we go and look for them. To avoid poor naming practices, we first document the names in a text document before we put them into Mariner which doubles the work.
2. While we're creating the attributes, Mariner saves each attribute one at a time to the database. To describe this process in a clearer light, if you had 10 attributes to make, you would make one in Mariner and click save, then wait 15 to 20 seconds for the program to save the attribute, then click new and make the next one. One would average about one attribute per minute using this process.

Overall, the process is long, tedious, and takes away a lot of time from the user. The attribute creator I made in VBA allows the user to specify each attribute's specs in an excel document. This document then becomes the documentation for the user. After the documentation is completed, the user can click "Create Attributes in Mariner" and the program will open Mariner, put every attribute in quickly, and allow the user to address other needs.

Batch Save to the Database Process

Mariner keeps project statistics up to date. When a project summary is opened, the UI is updated with information from the database. However, the database only updates statistics that are given to it. That is to say that the database doesn't calculate statistics based off other statistics.

This problem manifests itself when large reports are created based off the information in the database. If the information has not been saved, then the database will present data with noticeable errors. Therefore, I created a batch process that goes into Mariner and saves all of the projects through the program and thereby updating the database with the correct information.

Overall, these two processes should increase user's effective time through the attribute creator and the accuracy of data through the batch save process.

Project Documentation

The creation of both pieces of the project presented a wealth of learning and understanding about VBA, HTML, and Mariner. Each presented a number of unique struggles, but in the end I realized that VBA in Excel is a great way to create something that you need to use right away, documented, and fast.

Attribute Creator

The first thing I created in the attribute creator was the UI. I used a large user form that contained a few buttons, text boxes, and one large list. The first thing was to get the needed information from the user: the attribute's name and type. Mariner allows for 23 different types of attributes, so I started with two types to keep it simple until I had the functionality working. The combo box of attributes pulls from a sheet that holds all of the attributes' names to populate itself. This I found later to be very convenient as I wanted to add more attributes and include certain properties to each type. I'll explain more about this when I talk about the "Set Attribute" button after a couple paragraphs.

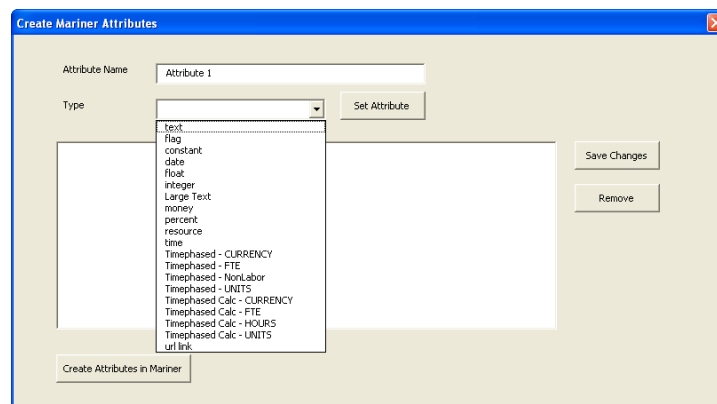


Figure 1 Type List

The next part consisted of populating the list in the user form and matching it to Excel document. The Excel document had two columns, one for the attribute name and the other for the type. The list within the user form had three names, one for the attribute name, type, and index number. When an attribute is created, the two lists are given the information to display. To learn how to add items to each list, I had to research object individually in order to successfully implement the function.

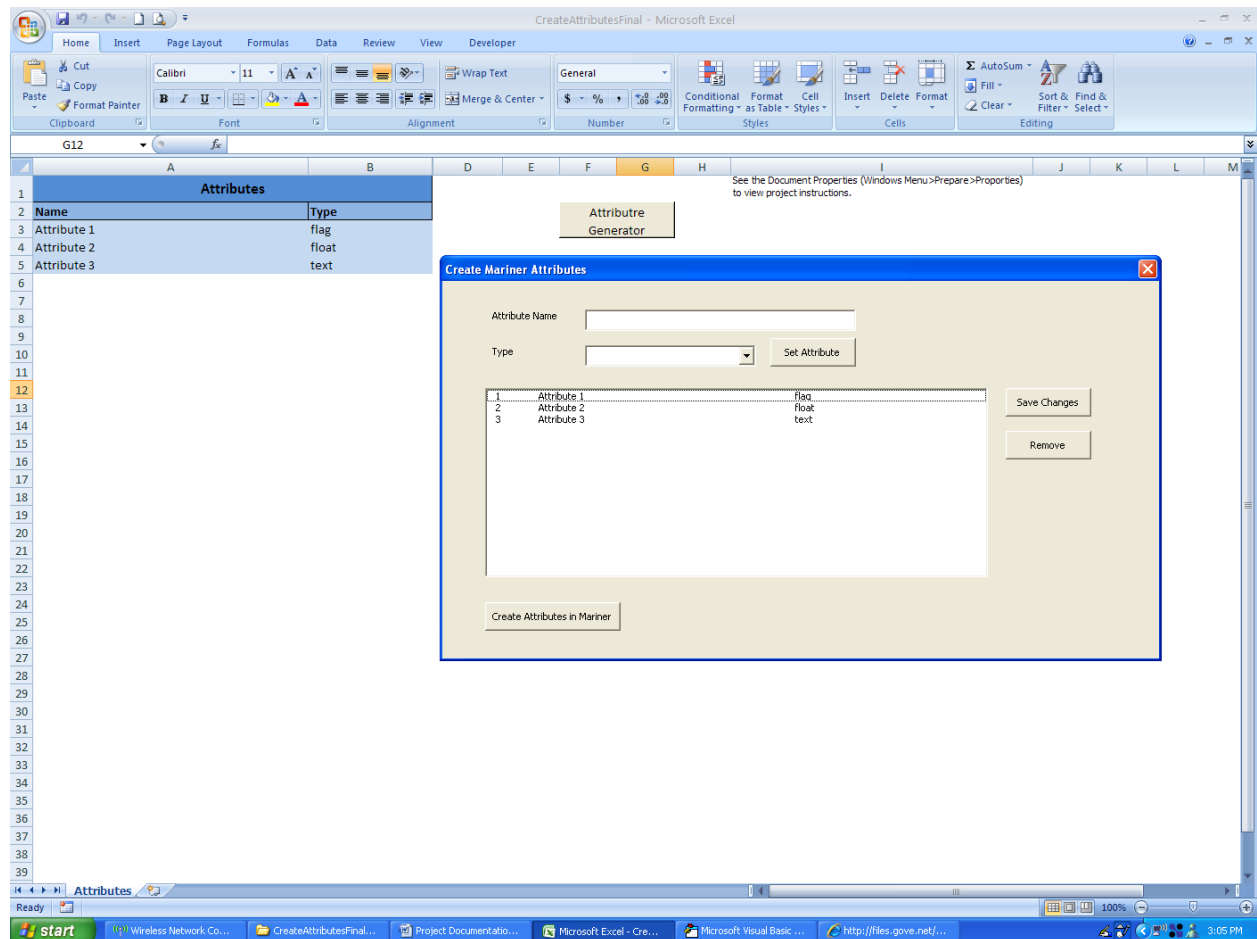


Figure 2 Displaying both identical lists

A feature I added to the user form list is that if the user clicked on one of the attributes, the attribute's information would be displayed in the text and combo box. This feature becomes apparently useful when the user is creating multiple attributes with similar names, which is a regular occurrence. For this to work, two different buttons were created on the user form.

The first button is a "Set Attribute" button. This adds the attribute to the list. If the user needs to use a similar name for another attribute, all they have to do is click on the one that is similar, edit the displayed information, and click "Set Attribute". A new attribute will be created in the list and the other attribute that was clicked on will be unchanged. To change an already created attribute, a "Save Attribute" button was created that serves that purpose.

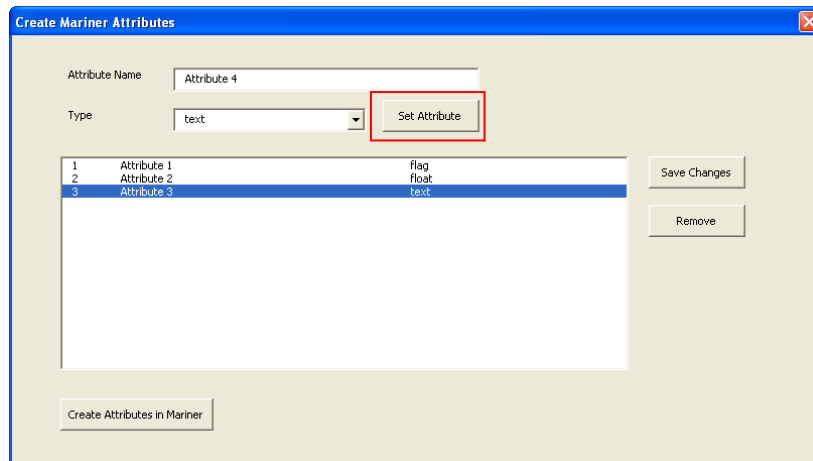


Figure 3 Showing ease to create attributes by using already created attributes as a template

The next button is the “Remove” button which removed the selected attribute from the list. As simple as this seems, two problems arose that required some VBA/Excel logic to make the button functional. The first problem arose when I tried to delete the name cell first followed by the type. The name cell is in column A and type in B. If I deleted the name in column A first, then Excel would move the type from B over to A. Then I would try to delete the type from column B, but since it was moved, I don’t delete anything and the type sits mockingly in column A. The simple change to delete B before A is very simple, but the logical understanding is noted.

The second problem came when the index numbers of the user form list became inconsistent after deleting an attribute. If I had attribute 1, 2, and 3, and I deleted 2, the index would read attribute 1 and 3 instead of attribute 1 and 2 (see table 1 and 2).

Table 1

	Name	Type
1	Service – Service ID	text
2	Service – Manger	text
3	Service – Function	text

Table 2 (after deleting attribute 2)

	Name	Type
1	Service – Service ID	text
3	Service – Function	text

The problem was solved using a refresh method that renumbered the list after an attribute was removed from the list.

After the buttons and input boxes were set, the next feature to implement was to update the user form list with attributes from the Excel document if any were already present. This takes place when the user form is opened. The Excel document is scanned for any existing attributes in the list and the user form list is populated.

Before talking about how I put the attributes into Mariner, I want to mention what I learned from the few hours I spent working on this user form. I learned that Excel has a very quick method for creating a UI and integrating it with the spreadsheets. If something must be made on the fly, either for actual use or even a prototype, Excel’s VBA implementation is a viable option.

To get these attributes into Mariner took some work. The code is short and brief, but the learning curve behind understanding the HTML is a little steep. I initially ran into a problem when I opened the “Attribute Manager” inside of Mariner and couldn’t find the text boxes in the HTML. I used a version of Firebug Lite for Internet Explorer that revealed that the boxes were in integrated frames. My first thought was confusion. *What are integrated frames and why can’t I access the code inside of them?* I struggled with the concept for a while, but eventually received information about a Microsoft site that explained the properties of an HTML document. From the documentation, I learned about integrated frames (aka iframes) and how an HTML documents object can access them through VBA. I played with the code for a couple of hours until I learned the ins and outs of iframes. The original HTML of the page has three iframes. The first one is the one that contained the desired data. However, when I asked for the first iframes object count it gave me a much lower number than I expected. In fact, the HTML itself was no more than 50 lines and I expected at least a couple hundred. After some further investigation, I found that this iframe and a few iframes imbedded inside of it. Again, the first iframe in the array was the desired one. Once I got into that one, I was finally able to access the text boxes, check boxes, and drop down menus.

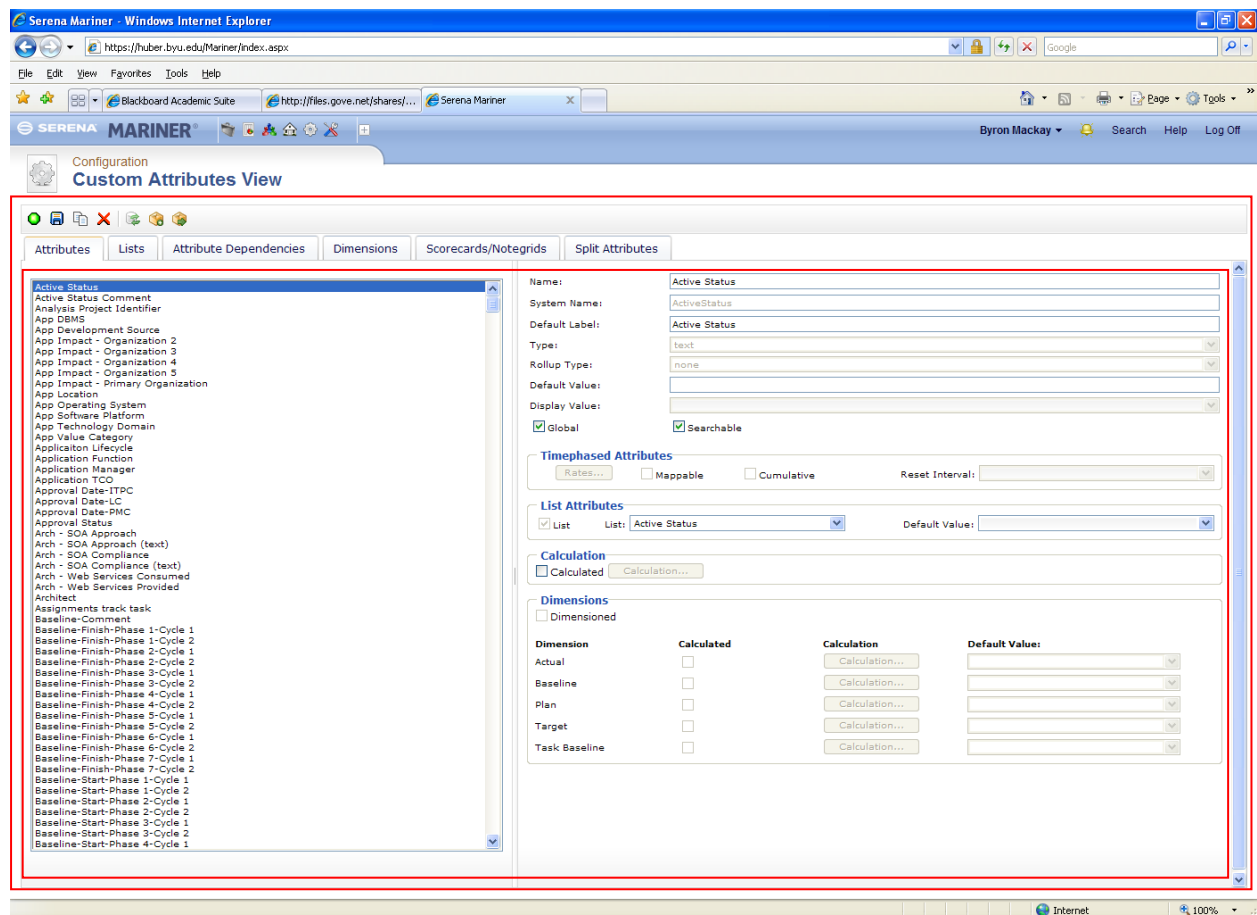
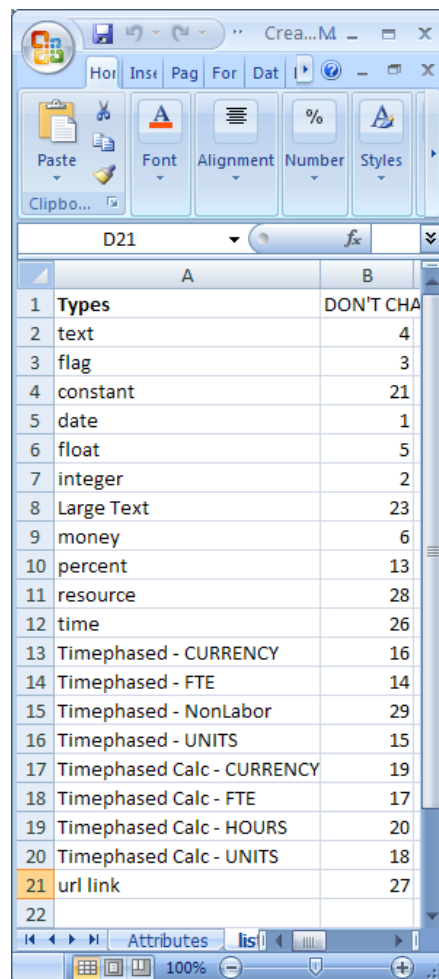


Figure 4 Iframes in iframes within HTML

The text box and check boxes gave me little trouble. The combo box, on the other hand, was obviously a victim of poor planning and programming. My initial thought was that since the user form has the same

order as the combo box in Mariner, then the index in the user form would be the same as the combo box in Mariner. To my surprise, not only were the indexes not the same, but the index numbers in Mariner ranged from 1 to 28 (only 23 types exist). Not only did the numbers not match, but the index number followed no numerical order. For example, the first three index numbers in Mariner's combo box are 4, 3, and 21. Once I delved through the mess of sorting the numbers around, I used another sheet in Excel to hold the Mariner index numbers. With the list available, I run a VLookup function before putting the type into Mariner that gives the correct index to identify.



	A	B
1	Types	DON'T CHA
2	text	4
3	flag	3
4	constant	21
5	date	1
6	float	5
7	integer	2
8	Large Text	23
9	money	6
10	percent	13
11	resource	28
12	time	26
13	Timephased - CURRENCY	16
14	Timephased - FTE	14
15	Timephased - NonLabor	29
16	Timephased - UNITS	15
17	Timephased Calc - CURRENCY	19
18	Timephased Calc - FTE	17
19	Timephased Calc - HOURS	20
20	Timephased Calc - UNITS	18
21	url link	27
22		

Figure 5 Mariner's numbers for type combo box index in Excel

The final piece of this part of the system consisted of a window that directed the user to perform certain actions. Mariner is password protected and I didn't want to put any password in the code. So when the "Create Attribute in Mariner" button is clicked, IE is opened to the website, but an Excel user form appears that asks the user to log in to Mariner and navigate to the Attribute Manager. Upon doing so, the user clicks OK and the program continues to run.

Batch Save Process

Not as beautiful but just as effective as the attribute creator, the batch save process enters Mariner and saves the projects one by one to the database. This seems unnecessary, but the database only saves data that it is given. Furthermore, the database doesn't calculate new values for data dependent on other data.

For example, let's say the project budget is \$100,000 and \$50,000 has been used up, so \$50,000 is left over. All three of these statistics are inside the database. A person inputs into mariner a purchase of \$20,000. The purchase in the database is recorded, but the money left over still reads \$50,000. When the person opens the project summary again, the program takes the information from the database and processes it. The program notices the new purchase for \$20,000 and calculates the new amount left over of \$30,000. After the person completes what they intended to do with the summary, they click save. The program sends all of the changed attributes to the database. Now the database reads that \$30,000 is left over.

The save process begins at midnight and opens Mariner in IE. Once logged in, the program must find all the projects within a given folder. The folder is found within a large directory and finding the right folder in the HTML can be tedious. After some fidgeting with the HTML, I realized that I didn't have to open each folder individually and visually find the folder I was looking for. The page always contains all of the folders in a list. Instead of searching for the folder by looking through its parent folders, I could simply go directly to the desired one. This saved a lot of tedious testing before execution. If I had to go through each parent folder, I would have had to test to see if it was open first, and then click it if it wasn't.

Once the folder is open, the program clicks on any file inside of the folder that has a project icon. This was done using the images list that can be retrieved through the HTML document object. When a project is identified, the program clicks on the icon thus opening the project's page. Once inside the page, an edit must occur in order for the project to be saved to the database. A checkbox is conveniently set as a dirty attribute and can be clicked before pressing save. This saves the information to the database. Once all of the projects saved, the database is ready to be accessed for reports.

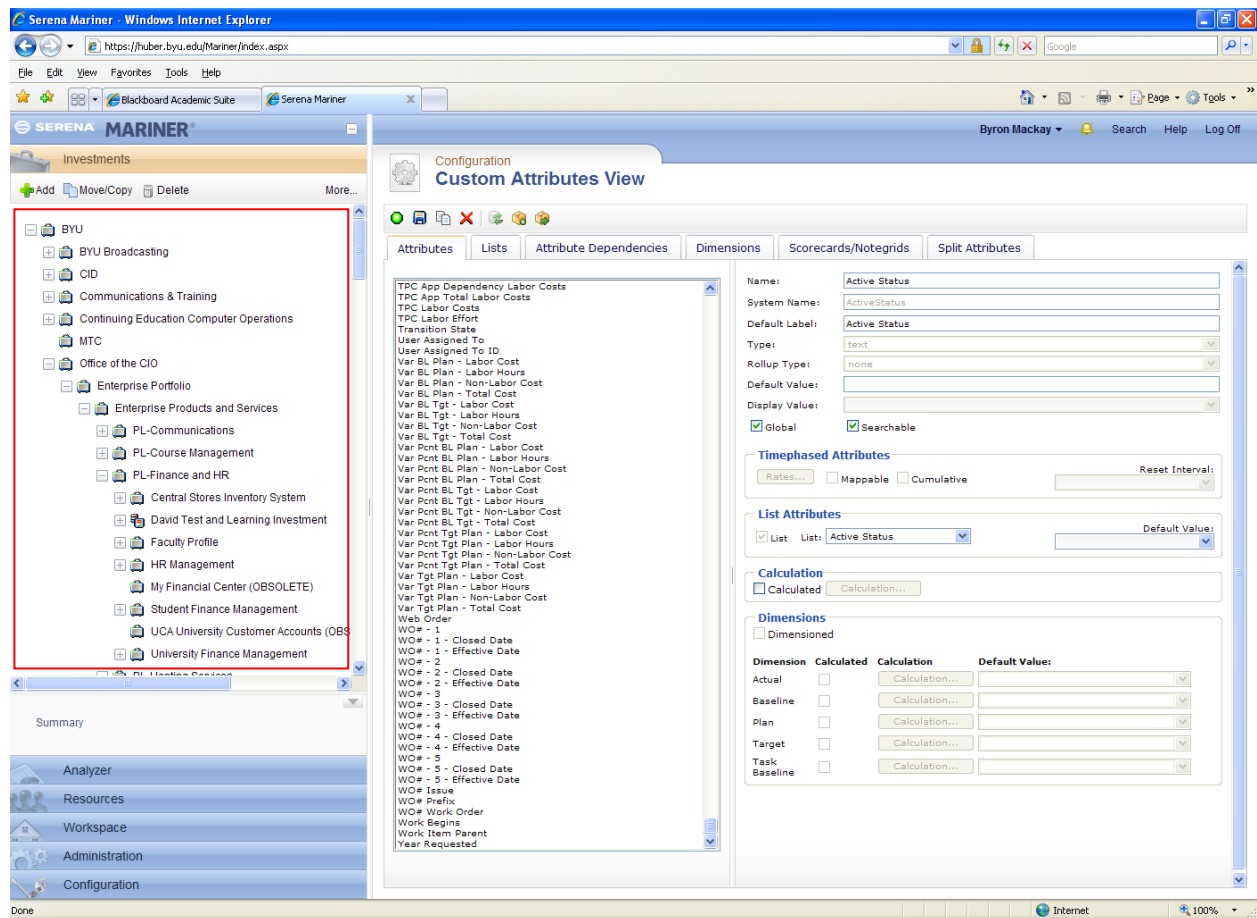


Figure 6 File structure inside of Mariner

Learning and Conceptual Difficulties

This project taught me a lot about programming, both within VBA and without. Though some of the newer concepts came easily to me, a majority of them took a lot of time and effort to discover and implement. The things I learned are as follows:

- How to create a VBA user form
- The inherited functionality of various user form objects
- The document object's functions and capabilities
- How to control Internet Explorer from VBA

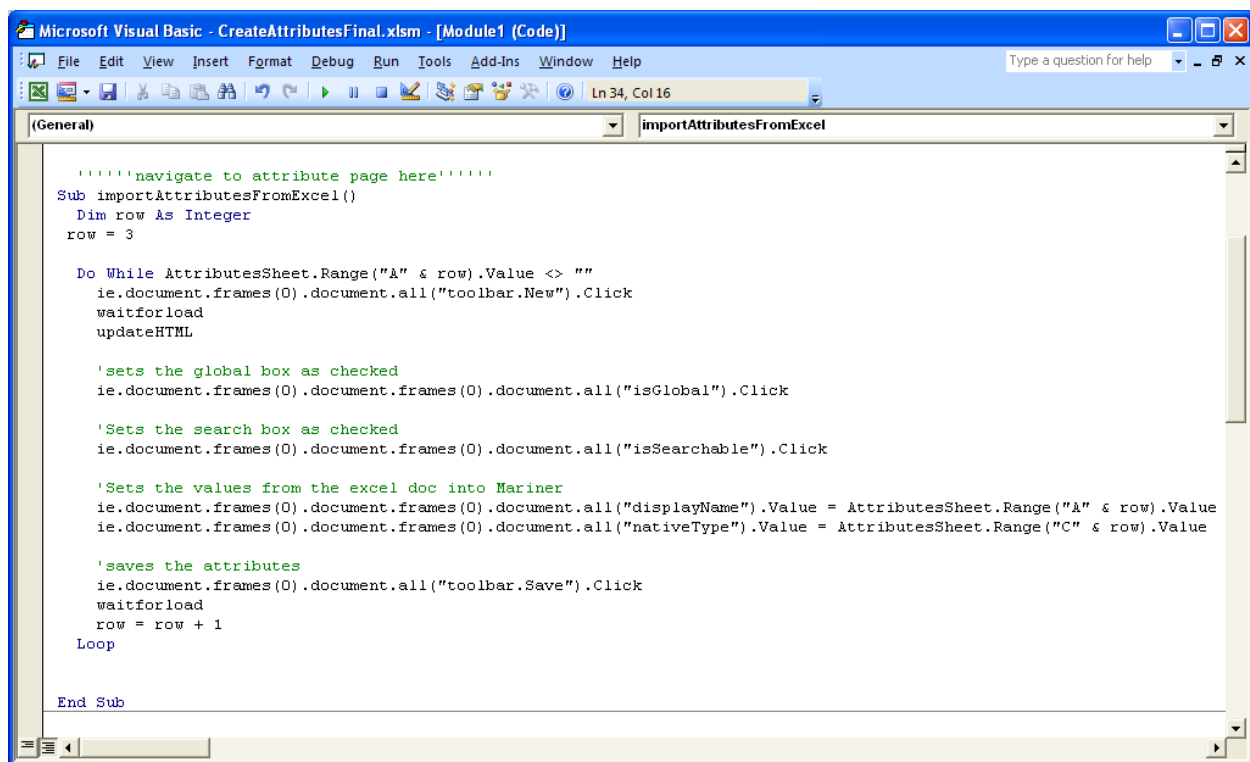
Before this class, I had no idea that Excel could create user forms. Now with some experience I have learned that user forms can be used for a variety of reasons, one in particular being to design a spreadsheet appropriately so it can be used by another program. I also appreciated how simple it is to create a user form. My form for this project took me about four to five hours to make with all of its

functionality and time it took to research certain pieces included. If I had to make this again, it would probably take me half the time.

Within the user form's objects, I learned what capabilities exist and which capabilities I needed to implement. For example, it seems unwise for a list object not to have a function to automatically number each row. If there is one, then I couldn't find it. Nevertheless, it wasn't difficult to create my own numbering system within the program.

I also learned a great deal about the document object in VBA. This object is very useful to edit HTML documents online. This took the most time and gave me the most headaches in this project. I had a hard time understanding how the object worked and played with it a lot until I found the right syntax. Now the code is only a few lines long, but the process to discover the code is probably worth a thousand lines.

The biggest resource in finding the correct syntax was Microsoft's documentation. Even with the company's own documentation, I found many functions to be useless and through trial and error I found a working solution.

The image is a screenshot of the Microsoft Visual Basic editor window. The title bar reads "Microsoft Visual Basic - CreateAttributesFinal.xlsm - [Module1 (Code)]". The menu bar includes File, Edit, View, Insert, Format, Debug, Run, Tools, Add-Ins, Window, and Help. A toolbar with various icons is located below the menu bar. The status bar at the bottom indicates "Ln 34, Col 16". The main code area shows a subprocedure named "importAttributesFromExcel". The code includes comments in green and VBA syntax in blue and black. It starts with a comment "navigate to attribute page here", followed by a loop that iterates through rows of an Excel sheet, clicking buttons to set global and search attributes, and then saving the attributes. The code ends with "End Sub".

```
''''''navigate to attribute page here''''''
Sub importAttributesFromExcel()
    Dim row As Integer
    row = 3

    Do While AttributesSheet.Range("A" & row).Value <> ""
        ie.document.frames(0).document.all("toolbar.New").Click
        waitforload
        updateHTML

        'sets the global box as checked
        ie.document.frames(0).document.frames(0).document.all("isGlobal").Click

        'Sets the search box as checked
        ie.document.frames(0).document.frames(0).document.all("isSearchable").Click

        'Sets the values from the excel doc into Mariner
        ie.document.frames(0).document.frames(0).document.all("displayName").Value = AttributesSheet.Range("A" & row).Value
        ie.document.frames(0).document.frames(0).document.all("nativeType").Value = AttributesSheet.Range("C" & row).Value

        'saves the attributes
        ie.document.frames(0).document.all("toolbar.Save").Click
        waitforload
        row = row + 1
    Loop

End Sub
```

Figure 7 Short code to change attributes

Finally, the last thing I learned was with the document object and how to manipulate a web site through VBA. I learned how to input values into textboxes, find specific information, and change the HTML code itself. The ability to change the HTML code is a little scary. I can think of only a few honest reasons and a bunch of dishonest reasons for why you'd want to change the HTML code.

Ultimately, this class has been a success. I have learned the reach of Excel and what it can do. I also have a solid foundation in VBA which has given me the confidence to be able to implement it in the real world. A personal evidence of this occurred when my boss asked me to modify the color scheme of a report he had in Excel. Within a few minutes I was able to turn the report back to him with the new colors he requested. When I came back, he said, "Wow! That was really fast."