

Executive Summary

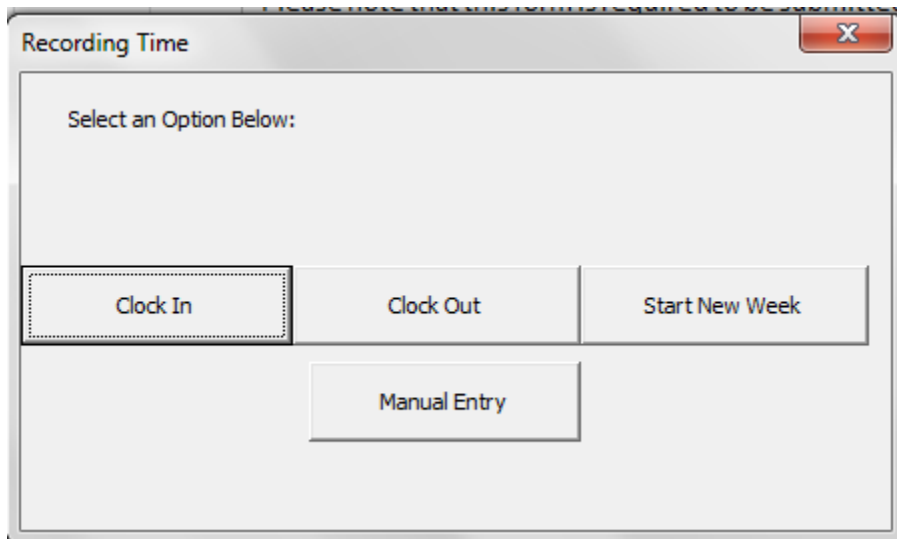
I work as a contractor for a national consulting firm called Connor Group. Over the last year, I have had two major responsibilities. First, I was in charge of managing research activities on a variety of both public and proprietary data related to initial public offerings (IPOs), where I did some of the research personally and managed other interns who also did some of the research. Additionally, I was put in charge of recruiting and managing new interns as they came in.

My project contained two major parts. First, Connor Group's HR has a specific Excel template for employees to report their hours weekly, but keeping track of time is left up to employee discretion. My first task was to develop a robust timecard that would allow interns to keep track of their hours on a week-to-week basis. My second task was to take a variety of different time-inefficient processes from the research projects and automate them, so our interns can focus on the parts of the studies that we are actually hiring them to do, rather than on, for example, data entry.

Implementation Documentation

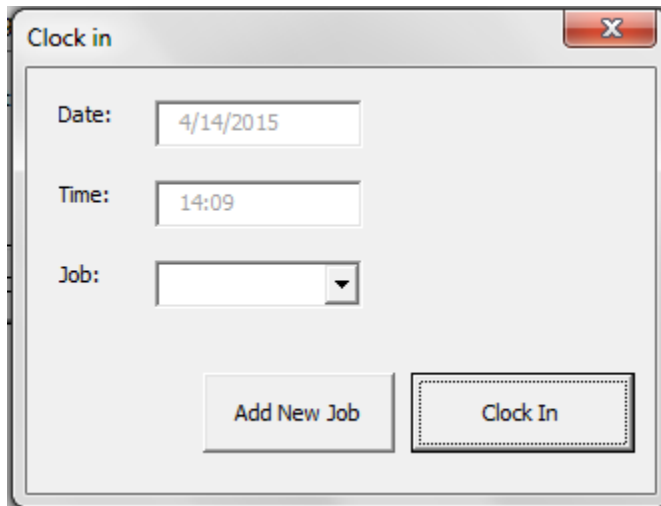
Timecard

The timecard was the first part of the project. I initially created all of the timecard functionalities in a user form, the final version of which is as follows:

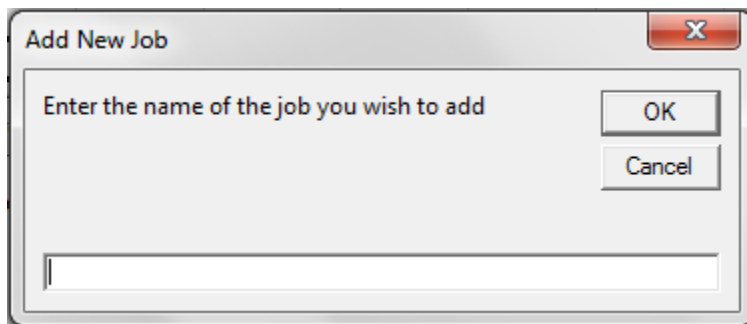
The image shows a screenshot of a user form titled "Recording Time". The form has a light gray background and a standard Windows-style title bar with a close button (X) in the top right corner. Inside the form, the text "Select an Option Below:" is displayed. Below this text, there are four buttons arranged in two rows. The top row contains three buttons: "Clock In", "Clock Out", and "Start New Week". The bottom row contains one button: "Manual Entry". The "Clock In" button is highlighted with a dashed border.

I later included all of these functions on the ribbon, but the user form is still available as well. Each of these buttons interacts with a table on the spreadsheet. The "Clock In" button opens a second form, as shown below. This form automatically includes the current date and time, and allows the user to either select a project or click a button to create a new project. Projects are stored on a very hidden sheet, so jobs that are added can be used later. I also implemented a

control that prevents users from creating “blank” projects. If a user attempts to clock in without providing a project, they receive a brief message informing them that they must select a project.



The 'Clock in' dialog box has a title bar with a close button (X). Inside, there are three input fields: 'Date:' with the value '4/14/2015', 'Time:' with the value '14:09', and 'Job:' with a dropdown arrow. Below these fields are two buttons: 'Add New Job' and 'Clock In'.



The 'Add New Job' dialog box has a title bar with a close button (X). Inside, there is a text input field with the placeholder text 'Enter the name of the job you wish to add'. To the right of the input field are two buttons: 'OK' and 'Cancel'.

If a user has “clocked in” to a job already (if there is a space on the time table that has a clock in time and no clock out time), they are not allowed to open the clock in form, and receive a message that they must first clock out of all other jobs before clocking in again.

The clocking out functionality is the simplest both in implementation and in coding. Clock out simply finds the first empty row, backs up one row, and enters the desired time to clock out, unless that cell is not empty. If something is in the cell, the user gets a reminder message that he/she has already clocked out.

Finally, the most complex functionality is provided by the “Start New Week” button. Starting a new week causes three things to happen. First, there is a control to make sure that no row has partial data. Second, the timecard form that must be submitted each week is generated, and finally a sheet is created for the new week.

The really interesting part is the creation and filling out of the timecard form. I began writing code that would recreate the form, but I quickly realized that this was wasting time-instead, I simply included the form itself as a very hidden sheet (so average users won’t make any changes that would interfere with HR’s processes on their end). When the program runs, it makes a pivot

table out of this week's time, then creates a new workbook that includes copies all of the required sheet. The pivot table summarizes time by day and project, and I wrote a subprocedure that performed a function similar to a match function. This sub drops the time worked into the cell that has the corresponding date across the top, and the corresponding project along the left. Once this is done, the pivot table sheet is deleted, and a new worksheet with a blank table is created to accept next week's data.

Some interesting tidbits that I included in the timecard: first, Connor Group ends its weeks on Saturday. The timesheet that HR uses is set up in such a way that my code will throw an error if the timecard includes dates that are part of more than one week, or if the timecard is submitted later than the week after the time was worked. However, that error is not a bug; it is a feature! If the employee submits time that "straddles" a Saturday, I didn't want to make any assumptions—instead, they need to figure out the problem with the time they've recorded. If they are submitting a timecard for work more than a week late, that's a serious problem with HR, and I don't want the code to make any assumptions in that case either. If either of those things happens, an error handler returns a strongly worded message that the employee needs to straighten out their data, and suggests these two possibilities as what might have gone wrong. I expect that this will be the case for 95%+ of erroneously submitted timecards.

IPO Metrics

IPO Metrics is a study that deals with metrics for assessing the IPO market. I included a small subset of the data, but the real dataset is huge. Significant time is put into looking up and entering data. One piece of data in particular takes significantly more time than the others: first-day stock prices. This functionality is accessed by a button (on the worksheet, not the ribbon), which asks which rows to find data for, then iterates through those rows, finding the needed data online.

This problem is just enough different from the typical "latest stock price" problem that we worked in class to make it interesting. Because I want the first available closing price, not the latest price, I can't just import a page and get the quote. Also, simply importing the table of all the stock prices doesn't work, because if a stock has been trading for long enough, they truncate the table. There may have been data sources available that would do what I wanted, but I knew how Yahoo Finance worked already. I found a URL that takes the ticker symbol of the company and provides historical prices. This page allows inserting a start date and an end date to generate a table of closing prices in a specific range. I used the agent class module we used in class, and manipulated the date range to only include the first day's stock price. Then I simply do some searching to get the data I need. Since users are allowed to enter rows that may already have data, I included some code so it wouldn't overwrite values that are entered into the spreadsheet, but will instead put a red border around any cells that disagree with the number the macro found online. This way, we can use the code to double check the huge batch of previous data.

I did not include this functionality on the ribbon, simply because not all of the people who receive these reports are Excel-savvy. I don't want them to run this code and throw errors that they shouldn't be seeing and get nervous about what they have just downloaded onto their computer.

Comment Letter Links

Connor Group also analyzes comment letters that companies receive from the SEC. Details related to comment letters are downloaded from a website, with a ton of excess data. Fortunately, the data always comes in the same format.

This part of the project is rather simple, and includes only two tools. These tools are both on the ribbon, which I felt comfortable with because only employees familiar with Excel will ever see this report. The data is brought in, and I wrote a quick macro that generates a pivot table from the data, as well as a second table that includes only the links to the comment letters in question. Employees can then filter the pivot table to include only the companies whose letters they are interested in reviewing, and click a second button that resizes the comment letter table so they don't have to sift through mounds of URLs to find the one they want to review.

CFO Compensation

Finally, Connor Group performs a study of CFO compensation in companies that complete an IPO, as a sort of incentive to help pitch IPOs to potential clients. Two tools were necessary for this study. First, sometimes CFOs don't work at the same company for the full year before and after the IPO. In order to make our data more comparable, my senior manager requested that I annualize the data of CFOs, as necessary. Again, I've only included a subset of the data, but the original study is massive, and prior interns have annualized no data. I wrote a quick macro that determines whether a CFO was present for 12 months in both years 1 and 2. If the CFO was present for part of both years, but absent for some portion of one or both years, then the macro does some quick math to determine what the annualized numbers should be, inserts those numbers into the appropriate columns, then indicates that data was annualized in column 2. This will allow us to see which numbers are estimates, in case we ever want to pull these numbers out.

Secondly, this study needs to be formatted regularly so it only shows the information that the partners want to see, and in a pretty format. My second macro "hides our work," so to speak. It filters out the rows of data we don't want included in the final summary metrics by only including companies that say "included" or "annualized" in column B. Next, it copies the filtered data onto the worksheet that is connected to the summary tables (where some Excel formulas summarize the data from the included companies). Finally, on each of the summary table tabs, the macro uses the replace method of the ranges that include the summary tables to resize the formulas to include the new data on the worksheet with the included data.

Similar to the IPO Metrics study, I had concerns about allowing the partners who receive the report to fiddle with the extra buttons on the ribbon, so both tools are called from buttons within the workbook, on a sheet that is hidden before the workbook is passed to the partners.

Discussion of Learning and Conceptual Difficulties Encountered

I had some difficulties with the agent class module. In the IPO metrics study, I had some trouble with manipulating the date ranges on Yahoo Finance. For some reason, the beginning dates responded on my first try, and I could manipulate them to say whatever I want, but the ending dates would not respond to my code. I tried manipulating them directly by changing their value, I tried to set them as tags and manipulate their values, and even though I had their IDs and their names, nothing worked. However, I was able to work around it by setting those elements equal to the start date parameters. In fact, I discovered that Yahoo Finance automatically uses the first day the stock was offered as the start date in the date range (even when it truncates that data in the table it is presenting), so in the end I was able to simply set the two ranges equal to one another and move on.

I had another problem with the agent class, which I never did solve. Unfortunately, I didn't read the instructions carefully enough, so I never asked for help from Professor Allen on this. There is a proprietary website from which we get a large portion of our data for CFO comp and IPO metrics, and I wanted to automate the download of that data. Similar to the Yahoo Finance problem, I needed to manipulate some date ranges, which I was actually able to do just fine this time, but when I tried to use VBA to click the button to bring me only the data in the date range I specified, (or the button to download the data, for that matter), nothing worked. The only thing different that I could see was that the buttons were connected to some Javascript on the site, but I couldn't see any reason that I shouldn't have been able to order the macro to click on the button anyway (I also tried submitting the page as a form, but the page didn't actually have a form to submit). Ultimately, I think the project was still broad enough in scope, but I will probably keep working on this problem after the semester is over, because my senior manager would really like to see this part of the projects automated. For now, I left a button on the spreadsheet that links to what I have so far. It requires a username and password, though, which I'm not allowed to disclose anyway. I did learn a lot from struggling with this, and I even wrote some cool code. I was having trouble getting the code to click on an arrow to open a form, then wait until the form was available to fill out the boxes and submit the form. The code refused to wait, until I came up with a clever "Do until..." loop. I had the code attempt to set the variable "tag," as one of the elements on the form, and repeat that until the tag was successfully assigned to the variable. This worked perfectly.

The timecard seemed like a really simple idea at first, but it ended up taking much longer than expected. My final solution required some creative dynamic arrays. It was surprisingly difficult to get each of the projects included on a timecard into an array once and only once, and it took me some time to think of declaring a public dynamic array, then redimming it (with a preserve

statement) as I added each new project. I also had to write a function that compared the name of each project to all of the projects in subsequent rows, to determine if this was the last time a project name was used. Technically, my solution only includes the last time a project name is used, but as long as it gets all of the projects into the array properly, I didn't mind.

Another quirk of the timecard that took me some time to figure out was dealing with leftover time. Connor Group HR requires that time be reported in EXACTLY increments of a quarter-hour, which must be represented as .25 (the ultimate timecard to be submitted recognizes hours as whole numbers). This means that frequently, bits of time that don't add up to 15 minutes get discarded, so I included a catchall "leftovers," variable that picks up the time that doesn't get recorded. At the end of the process, if the macro picks up enough leftovers, it adds that time to Saturday's time, to make sure the employees aren't shortchanged for working a few minutes here and there.

Finally, I completed most of the pivot table part of the timecard before we discussed pivot tables in class, so creating and formatting the pivot tables correctly required a lot of research and study, but I got it to work on my own, the weekend before we discussed it in class.

Disclaimer

I did not end up getting any assistance on the project (except from publically-available internet sources, which mostly provided ideas rather than actual code). I did not knowingly copy code from any source, online or otherwise. I also did not work with any other students on my project. Everything you see on the VBA side is my own work (with obvious exceptions for things like the agent class module that Dr. Allen showed us in class).