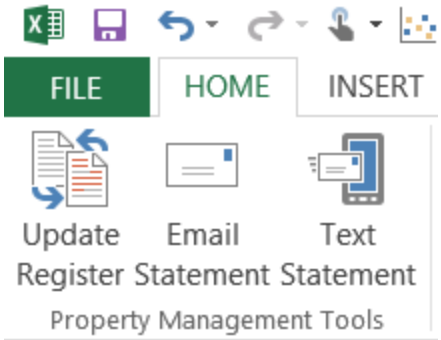
Property Management Tools

USING EXCEL VBA TO AUTOMATE RENTAL PROPERTY ACCOUNTING



Ben Chapman

9 April 2016

MBA 614 | SPREADSHEET AUTOMATION | WINTER 2016

Executive Summary

Business Background

One of my businesses owns and manages residential rental real estate. Generally, the tenants deposit their monthly rent payments directly into a special "deposit account" I have set up at the bank. In the past, I have manually checked the deposit account to verify the dates and amounts of monthly rent payments. Then I have manually transferred this information into a basic Excel file and calculated relevant late fees as necessary. Finally, I have taken time to personally contact tenants as necessary regarding past due amounts, upcoming payments, etc.

Problem

Manually looking up tenant deposits, calculating rent and late fee charges, and personally contacting tenants is tedious, time consuming, and, while necessary, adds relatively little value to the business. In our technologically capable, but extraordinarily hectic environment it is essential to leverage the abilities of technology to reduce the burden of these types of repetitive, low value-add tasks. Time saved from automating this lookup, recording, and contacting process can be better spent finding new deals, evaluating opportunities, and developing business strategy.

Solution: System Overview

This tool automates the previously manual rental property accounting process in three ways:

1. The Update Register tool automatically updates the three primary components of the Property Account Register:
    a. Monthly Rent Charges—the code automatically calculates monthly rent charges due based on the difference between the current system time and past records already recorded.
    b. Rent Deposits/Payments—by automatically controlling Internet Explorer, the sub procedure logs into the online deposit/payment account, accesses the account history, and captures payments not yet recorded on the register.
    c. Late Fees—with all the charge and deposit data entered into the register, an algorithm is then able to calculate contractual late fees based on account balances, deposits made, and payment due dates.
2. The Email Statement code emails an auto-generated account statement as a PDF file attachment to an auto-generated email message. To accomplish this, the algorithm opens an Excel template, personalizes it for the tenant, and adds the relevant charges, credits, and account balance based on the date range specified by the user. Then, a personalized email is generated using a text file template that is auto-filled programmatically. Finally, a PDF copy of the Excel statement is attached and the outgoing message is sent.
3. The Text Statement procedure sends an auto-generated text message to the tenant's mobile phone. This personalized message includes historical deposit information, current balance credits or amounts due, and customized messages depending on whether the account is current or delinquent.

Occasionally, tenants need reminders to keep the importance of paying their rent on time at the top of their minds. Beyond automating time consuming manual operations, this tool offers the additional benefit of communicating regularly with tenants to help them keep their rent obligations as a top priority.

Implementation

General Use Case Workflow

While the code is robust enough to handle many additional use cases (e.g. downloading up to six months of previous payment history, specifying date ranges for what account statement line items to include, etc.), the intended general workflow for a property management agent using this system is as follows:

1. Each month the property manager downloads payments received and calculates payments due by clicking on the "Update Register" button conveniently located on the "Home" tab ribbon in the "Property Management Tools" group.
2. Next, the property manager clicks the "Email Statement" ribbon button to generate the tenant's monthly account statement and send it to them via email as a PDF file attachment.
3. Finally, the property manager has the option of contacting the tenant via text message (which in my experience tends to be the most reliable way of contacting them).

Detailed Programmatic Workflow

This project consists of three buttons conveniently placed on the Home ribbon in a "Property Management Tools" group. As seen in Figure 1, these buttons are as follows:

- Update Register
- Email Statement
- Text Statement

I created these buttons with customUI XML using the "Custom UI Editor For Microsoft Office" (see Figure 2). These buttons drive three primary sub procedures:

- updateRegister
- emailStatement
- textStatement

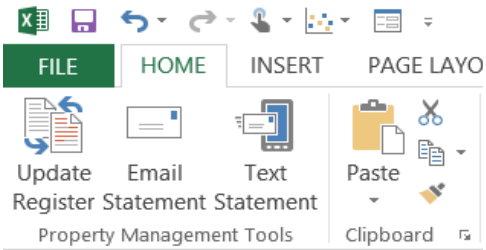


Figure 1. The Property Management Tools User Interface Ribbon.

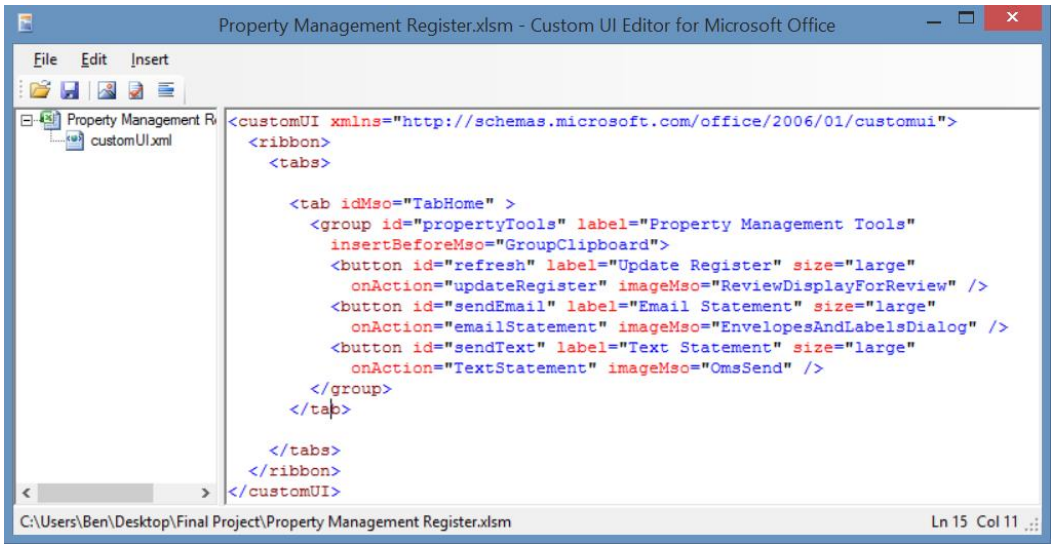


Figure 2. Custom UI Editor and Ribbon Button XML Code.

Each of these sub procedures and their supporting functions and forms are described in detail below.

Update Register

The updateRegister sub procedure (see full annotated code in Exhibit 1 of the Appendix) begins by finding the row of the last recorded "deposit" line item, the last recorded "charge" line item, and the next line entry (i.e. the blank row after the last recorded line item). Figure 3 shows the Property Account Register layout with these three rows highlighted.

Valley Rentals

Property Address:	**469 Riverside Dr.**	Tenant:	Chad Davis
	Apple Valley, CA 92307	Phone:	760-242-5885
Monthly Rent:	$1,100	Cel:	385-671-9281
Late Fee (after 5th):	8%	Email:	byuvbata@gmail.com
Bank Account Webpage:	www.wellsfargo.com		
Deposit Account Number:	7244934587		

Property Account Register

Date	Description	Deposit	Charge	Balance
1/1/2016	Balance Brought Forward			$0.00
1/1/2016	2016 January Rent		$1,100.00	-$1,100.00
1/6/2016	2016 January Late Fee		$88.00	-$1,188.00
2/1/2016	2016 February Rent		$1,100.00	-$2,288.00
2/4/2016	Rent Payment	$1,500.00		-$788.00
2/6/2016	2016 February Late Fee		$88.00	-$876.00
2/18/2016	Rent Payment	$876.00		$0.00
3/1/2016	2016 March Rent		$1,100.00	-$1,100.00
3/3/2016	Rent Payment Last Deposit Row	$1,100.00		$0.00
4/1/2016	2016 April Rent		$1,100.00	-$1,100.00
4/6/2016	2016 April Late Fee Last Charge Row		$88.00	-$1,188.00
	Next Transaction Row			

Figure 3. The Property Account Register Showing the Last Deposit Row (Red), the Last Charge Row (Green), and the Next Transaction Row (Purple).

Basic Error Handling of Initial Variable Values

The next code block takes care of error handling for various abnormal use case scenarios (e.g. initial settings for a new register, etc.). Figure 4 shows a couple examples of potential error catching input boxes. Incidentally, I've found error handling in Excel to be much more difficult than it needs to be due to the lack of "try/catch" functionality. And while currently programmed error handling doesn't deal with every possible scenario, it does provide very functional usability by catching the most common errors.

Update Rent Charge Line Items

With initial variables defined, it is now time to start updating the register. The first updating code block calculates and records rent payments that are due based on the date difference between the last charge line item listed on the account register and the current date. While the intent is that the register will be updated each month, the code will also update multiple months if some months have been missed.

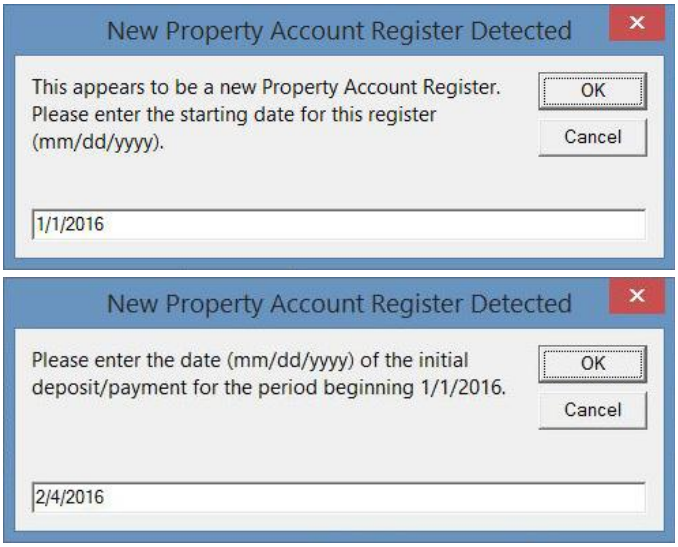


Figure 4. Error Catching Input Boxes.

Initiate Bank Login

The next code block launches a Wells Fargo Bank Login user form to capture bank login credentials (see Figure 5). It took quite a bit of time to design this professional looking form. I matched the colors and style to the Wells Fargo logo and the existing login form on the Wells Fargo banking website. If the user clicks cancel, the form unloads from memory and the sub procedure exits. Code is also included to ascertain whether valid login credentials are entered, e.g. leaving the username and password fields blank or providing incorrect login credentials triggers an "Input Error" message box (see example in Figure 6).

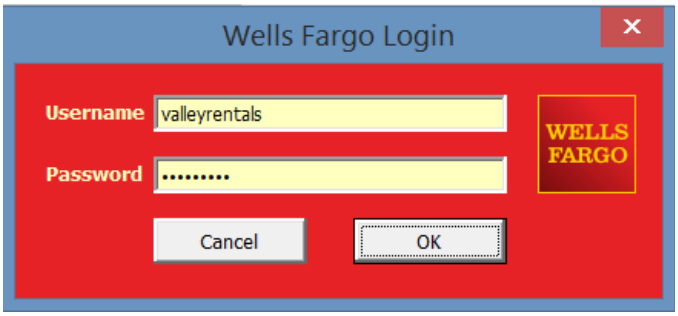


Figure 5. Wells Fargo Bank Login User Form.

Figure 6. Bank Login Error Message Box.

Automate Internet Explorer to Capture Deposit Transactions

The "agent" class is used next to login to Wells Fargo Bank's online banking system and navigate through multiple pages to access the transaction history for the correct account. Then the code cycles through each transaction until it finds the last one already entered into the account register. At this point, the "agent" steps back through the transactions capturing each new deposit (while ignoring any withdrawals) and entering them into the register. The existing "agent" class "moveTo" and "moveBackTo" functions didn't operate in a way that would efficiently accomplish this task so I wrote my own function in the "agent" class called "moveToPrevious" (see code in Exhibit 2 of the Appendix).

Once all the new deposits are recorded, the "agent" logs off the banking website and closes Internet Explorer. I should mention here that the online banking site defaults to showing only 90

days of transaction history. I added functionality to the code so that if it doesn't find a match for the last deposit transaction within the 90 day window it will automatically update the page to show 6 months of transaction history and try again. While in normal operating cases the user would update the sheet approximately monthly, the program will still work if several months slip by before updating is triggered.

Calculate Contractual Late Fees

The next block of code calculates late fees. Late fees are charged when the rent is not paid in full by the 5th of the month. It took a lot of thought to figure out how to write this multi-conditional algorithm. The first step sorts all the newly added records on the register by date. Then each line item's date and balance due are analyzed. Cases where the account balance immediately preceding the 5th are negative are then charged a monthly late fee. My initial attempt at this algorithm worked, but was a little clunky. After pondering the problem further, I re-wrote a much clearer version of the algorithm. The current "Revision 2" code accomplishes the same task, but is much cleaner and simpler. Finally, with the proper late fees entered into the register, the code sorts the register by date again to ensure that all rent charges, rent payments, and late fees are in chronological order. Then the code repaints the interior shading of the register lines to make up for the sorting that messed them all up.

writeRegisterLine Function

The updateRegister sub procedure writes to the Property Account Register in four different sections of the code: initial error handling, calculating rent charges, downloading new deposit information, and evaluating late fees. Initially, I coded each of these instances in separate instruction blocks. In my "Revision 2" edits, I optimized this substantially by writing a "writeRegisterLine" function that receives the relevant input arguments, writes the line to the register, and returns the updated "next transaction row" value (see code in Exhibit 3 of the Appendix). This significantly simplified the underlying code.

Email Statement

After updating the Property Account Register, the tool has two options for sending results and reminders directly to the tenant. The first of these is via an email message powered by the emailStatement sub procedure (see full annotated code in Exhibit 4 of the Appendix). The first code block (after variable declarations) sets initial variable values, opens the "account_statement_template.xlsx" template (the full template is displayed in Exhibit 7 of the Appendix), and attaches the "Property Management Register.xlsm" and the "account_statement_template.xlsx" spreadsheets to separate worksheet objects. Since this sub procedure references two different spreadsheets simultaneously, it is important to set each one as a separate and specific worksheet object that can easily be referenced programmatically.

Set Account Statement Transaction Start Date

The next code section sets the start date for line items to be included in the tenant's account statement. Initially, I hardcoded this to include all transactions beginning on the first of the month previous to the current date. In "Revision 2" (the current version), I added an input box where the user can select the desired date range. For ease of use, the input box defaults to the previously hardcoded value (see Figure 7).

Prepare Account Statement

The next step transfers the relevant "Bill To:" details (name, address, phone, email, etc.) from the internal company Property Account Register to the tenant's account statement. Then the code calculates the starting row in the company register based on the user specified start date and transfers each charge and credit/payment row within the relevant date range to the tenant's account statement (see completed statement in Figure 8). The finalized statement is then saved as a PDF file and the email attachment path is evaluated and stored.

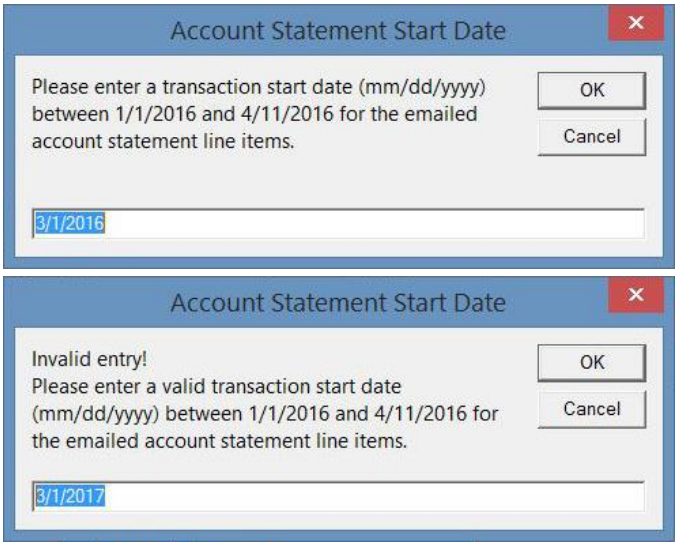


Figure 7. Input Box (and Error Handler) to Set Tenant's Account Statement Start Date

Valley Rentals
357 Corwin Rd
Apple Valley, CA 92307

Statement

April 12, 2016

Bill To:
Chad Davis
469 Riverside Dr.
Apple Valley, CA 92307
760-242-5885
byuvbata@gmail.com

Account Summary	
Past Due Balance	-
Credits	1,100.00
New Charges	2,288.00
Total Balance Due	**1,188.00**

Date	Description	Charges	Credits	Balance
3/1/2016	Balance Brought Forward			$0.00
3/1/2016	2016 March Rent	$1,100.00		$1,100.00
3/3/2016	Rent Payment		$1,100.00	$0.00
4/1/2016	2016 April Rent	$1,100.00		$1,100.00
4/6/2016	2016 April Late Fee	$88.00		$1,188.00

Figure 8. Completed Tenant Account Statement.

Get User Name and Gmail Login Credentials

The next task is to capture the property agent's name and the company's Gmail login credentials. This is accomplished through a professional looking user form that includes the Gmail logo (see Figure 9). This form also includes error handling programming to deal with blank entries. And like the Wells Fargo form, this form took quite a bit of time to design and tweak.

Prepare Email Message

The next code block opens the "email_message_template.txt" text file and reads it into memory. Then each of the coded fields in the email template (e.g. <YR>, <MO>, <RENTERNAME>, <PROPERTYADDRESS>, <AMOUNTDUE>, etc.) are replaced with their relevant values retrieved from the tenant's account statement. Finally, the code separates the subject and body sections of the message.

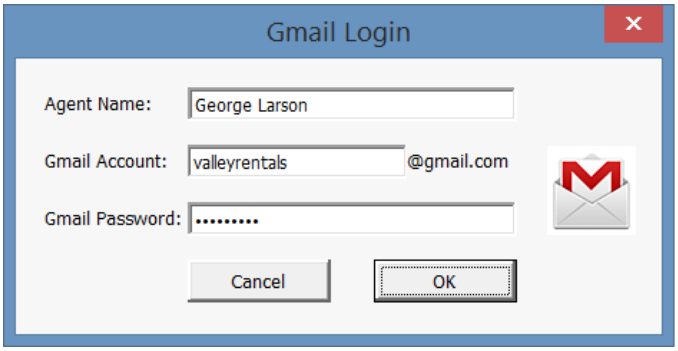


Figure 9. Agent Name and Gmail Login User Form.

Send Email and Display Confirmation/Failure

With all the sub elements prepared, it is finally time to send the completed email with its PDF attachment. This is accomplished by calling the sendGMail function (see code in Exhibit 5 of the Appendix) and sending it each of the input value arguments necessary to connect to the Gmail SMTP server, authenticate the user, and send the message complete with the attachment. The message boxes listed in Figure10 confirm whether the sendGMail function reports that the email sent successfully ("Email Confirmation") or failed ("Email Failed"). Finally, the code cleans things up by unloading the Gmail form from memory and closing (without saving) the account statement template excel file.

Figure 10. Email Confirmation/Failure Messages.

Text Statement

The final Property Management Tool button executes the textStatement sub procedure (see full annotated code in Exhibit 6 of the Appendix). This procedure does not create and attach an account statement PDF file. Instead, it prepares a specially designed concise text file and sends it as a text message to the tenant's cel. phone number. The major code blocks are discussed below.

Get User Name and Gmail Login Credentials

First, the code opens the Gmail Login user form to capture the property agent's name and the company's Gmail login credentials (see Figure 9). These values are stored in variables for later use. As mentioned previously, this user form also includes error handling code to deal with blank entries.

Prepare Text Message

After the user presses "OK" on the Gmail Login form, the code reads the "text_message_template.txt" template file into memory. Then the code prepares the message by replacing all the coded fields in the template (e.g. <RENTERNAME>, <LASTDEPAMOUNT>, <LASTDEPDATE>, <RENTAMOUNT>, <LATEFEE>, <ACCOUNTBALANCE>, etc.) with the appropriate values on the Property Account Register. Based on whether the tenant's account has a credit or a deficit, the <ACCOUNTBALANCE> field receives a customized credit vs. past due message. For example, if the account has a credit, the message reads, "Your account has a credit…. You do not have to make any payments." However, if the account is past due, the message displays, "Your account is past due! Please deposit payment…." Figure 11 shows the text message template and Figure 12 shows a completed text message for an account with a past due balance.

```
Rent Statement

<RENTERNAME>,
Your last deposit of
<LASTDEPAMOUNT> was received on
<LASTDEPDATE>.
Your monthly rent payment of
<RENTAMOUNT> is due on the first
of each month. An 8% late fee
(<LATEFEE>) is charged for
balances not paid by the 5th.
<ACCOUNTBALANCE>
We greatly appreciate it when you
pay your rent on time.
Sincerely,
<AGENTNAME>
Valley Rentals
760-946-9775
```

Figure 11. Text Message Template.

```
To: 3856719281@tmomail.net
Subject: Rent Statement
Body: Chad,
Your last deposit of $1,100.00
was received on 3/3/2016.
Your monthly rent payment of
$1,100 is due on the first of
each month. An 8% late fee
($88.00) is charged for balances
not paid by the 5th.
Your account is past due! Please
deposit payment of $1,188.00 to
avoid eviction proceedings.
We greatly appreciate it when you
pay your rent on time.
Sincerely,
George Larson
Valley Rentals
760-946-9775
```

Figure 12. Completed Text Message.

Create Text Message Email Address

The next block builds the "send to" email-to-SMS address by iterating through each digit in the cel. phone number worksheet cell and concatenating the numeric digits while skipping any non-number markers (e.g. parentheses, dashes, etc.) to form a numeric string. Then the mobile phone carrier's SMS gateway server address is concatenated to the phone number to create a complete email-to-SMS address. Currently, the program is hard-coded for T-Mobile (i.e. "@tmomail.net"). Future development will add functionality to allow the user to select from a list of carriers on the text message Gmail login form or have a box to record the cellphone carrier name on the Property Account Register itself.

Send Email and Display Confirmation/Failure

With the text message and "send to" address prepared, it's time to send the message. To do this, the sendGmail function is called and passed all the necessary input value arguments (see code in Exhibit 5 of the Appendix). Figure 13 shows examples of the ensuing message boxes that signal whether the message sent successfully ("Text Confirmation") or failed ("Text Failed"). Finally, the code cleans things up by unloading the Gmail form from memory.

Difficulties and Learning Opportunities

Understanding the Problem

The biggest initial conceptual difficulty was getting a clear picture of what I needed the program to do. Without a clear understanding of the problem the direction to the solution was undefined. As the Cheshire Cat says in Lewis Carroll's *Alice's Adventures in Wonderland*, if you don't know where you want to go, then it doesn't really matter what road you take. After struggling for clarity for a while, I finally sat down and had a "design meeting" with myself and carefully thought through the problem and what steps would create an effective solution. Then I outlined the project's major features, functionality, and general workflow. Finally, with a solid roadmap in place, I set out to write the actual code.

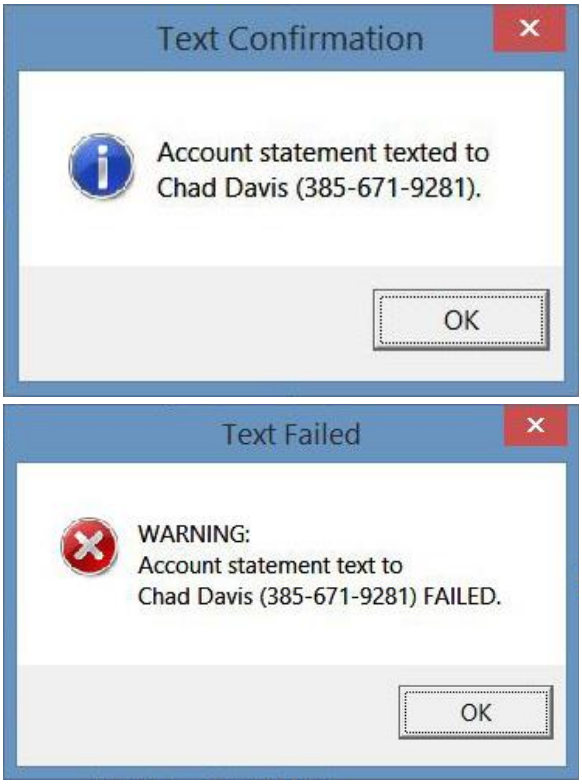


Figure 13. Text Confirmation/Failure Messages.

Referencing Multiple Workbooks Simultaneously

Referencing two workbooks at the same time was another challenge. In my first sub procedure (updateRegister), I take advantage of the fact that Excel "assumes" that I mean the "active sheet" when I'm reading from and writing to various cells. However, in my second procedure (emailStatement) this assumption quickly becomes invalid as I'm reading from and writing to two different worksheets in two different workbooks simultaneously: the "Register" sheet in the "Property Management Register.xlsm" workbook and the "Template" sheet in the "account_statement_template.xlsx" workbook. In this case, I couldn't write nonspecific "cells(row,column).value" code; rather, I had to specify the workbook and worksheet object each time to assure that the right cell on the right sheet in the right workbook was being accessed. Referencing the full workbook and worksheet object names in each line of code would have been tedious, cumbersome, and hard to read (e.g. `Workbooks("account_statement_template.xlsx").Sheets("Template").Cells(6, "A").Value = Workbooks("Property Management Register.xlsm").Sheets("Register").Cells(3, "E").Value`).

It was much clearer and less error prone to attach each worksheet to a worksheet object variable (i.e. "template" and "register" respectively) and then reference them accordingly (e.g. `template.Cells(6, "A").Value = register.Cells(3, "E").Value`).

Calculating and Recording Late Fees

Another conceptual challenge was figuring out how to calculate contractual late fees. Late fees are based on multiple conditions (i.e. transaction dates and account balances) that occur in a certain order. I had to establish the right order of events in the data and then find the various

conditions to ascertain whether a late fee should be charged. Moreover, I realized that the way to find the late fee conditions changes based on whether the algorithm is calculating a potential late fee for a past month or the current month. My initial attempt at this algorithm worked, but was rather unwieldy. During this initial coding session I kept finding myself thinking, "This isn't very good code." Finally I told myself, "I don't care if it is good code, I just need to get it to work." In retrospect I learned a lot from this mental process. Similar to writing a paper, I learned that in coding sometimes you need to just force yourself to press through the problem until you get something working, i.e. an initial draft. Then, after taking a break, visit the problem again and see if new insight helps you edit or re-write it better. As in writing good papers, editing and revising a second draft is much easier than trying to write a perfect first draft. In fact, I'm still thinking about this and some of my other algorithms and wondering whether there is a more elegant way of executing them.

Other Learning

I learned a lot working through this project. It was an enormous amount of work, but through the dozens of hours spent I gained valuable VBA coding and problem solving experience. Among many other things, I learned how to automatically save an Excel workbook as a PDF file, how to auto-sort a group of cells, and how to set the interior color and styles of cells. I also gained valuable experience designing, creating, and programming aesthetically pleasing user forms, spreadsheet templates, and custom ribbon buttons. Additionally, I gained a greater appreciation for the try/catch error handling methodology that is not available in Excel VBA as I dealt with the cumbersome "trial and error" science of error handling. Finally, by programming a new function for the "agent" class, I gained a deeper understanding of how the class works. For example, now I realize that the "agent" class downloads and stores the html source code from each page as a text string. This facilitates using basic string manipulation methods to find, extract, or replace the desired data.

Future Development

I've realized that software development is never really "done" as there are always additional features that can be developed. The code for this project is completely functional and operational, but there are several additions I would like to incorporate into future revisions. I've listed some of my future release ideas here.

Options for Multiple Online Banking Institutions

I have some other rental properties, each with their own deposit accounts at different banks. I'd like to develop this program further to have separate sheet tabs for each property with bank specific code that updates rent payments for each of them automatically. This will require additional login forms and totally different coding for the online banking interactions required at the different banking institutions.

Advanced Error Handling

In my "Revision 2" edits, I added a lot of error handling functionality. I'd like to look at this again and see if there is a more elegant way of dealing with potential problems and user errors. I also may endeavor to incorporate some of the error handling input boxes into the functionality of the login forms or create a form for manually entering transactions.

Mobile Carrier Options for Email to Text

I would also like to add functionality to select multiple mobile phone carriers within the textStatement procedure. Currently this is arbitrarily hardcoded for the T-Mobile carrier. I found a website (http://www.emailtextmessages.com) that lists the email-to-SMS gateways of dozens of carriers. I could composite this information onto a new worksheet that is searched based on the carrier value listed for the tenant on the Property Account Register worksheet or I could even have the "agent" search for the carrier on the website directly.

Conclusion

Assistance

I did not have any help designing, creating, or coding this project. With the exception of the "agent" class, the sendGMail function, and some initial ideas for the rent statement template, all code, forms, layout, design, etc. was my own invention. It would have been very difficult to successfully finish this project without the "agent" class and the sendGmail function, and I'm grateful to Dr. Allen for providing these resources for our use.

Summary

This was a huge stretch project for me. It took an enormous amount of time (well over the 20 hours specified), but in many ways was a great capstone project that gave me a chance to apply many of the tools and skills I have acquired in this class to solve a real business problem. To create these property management tools, I used user forms, event based programming, reading text files, opening workbooks, manipulating cells on worksheets, automating Internet Explorer, automatically saving files, manipulating text strings, and many, many more VBA elements. I also learned many new things about VBA through online research, macro recording, etc. as I pressed forward to complete the project.

Finally, since this project uses sensitive banking and personal information, I've stripped out the details and entered bogus example data in each of the fields. However, when the correct login credentials are supplied, the project works very well seamlessly downloading and compiling the real data to solve a real business problem I've had.

Appendix

Exhibit 1: Code for *updateRegister* Sub Procedure

```
Sub updateRegister(Optional control As IRibbonControl) 'Callback for refresh onAction
  Dim row As Long
  Dim lastChargeRow As Long
  Dim lastDepositRow As Long
  Dim transDate As String
  Dim transAmount As String
  Dim a As New agent
  Dim URL As String
  Dim i As Integer
  Dim r As Integer

  'set initial variable values
  row = Cells(Rows.Count, "A").End(xlUp).row + 1 'find row of next line entry
  lastChargeRow = Cells(Rows.Count, "F").End(xlUp).row 'find row of last rent charge
  lastDepositRow = Cells(Rows.Count, "E").End(xlUp).row 'find row of last rent payment

  'error handling of initial variable values
  If row = 13 Then
    row = writeRegisterLine(row, _
          linedate:=InputBox("This appears to be a new Property Account Register." & vbNewLine & _
                             "Please enter the starting date for this register (mm/dd/yyyy).", _
                             "New Property Account Register Detected"), _
          description:="Balance Brought Forward", _
          balance:=InputBox("Please enter the initial balance for this register.", _
                            "New Property Account Register Detected"))
    If Cells(13, "A").Value = "" Then Exit Sub
  End If
  If lastChargeRow = 12 Then 'if no charges listed on register then insert the initial charge
    row = writeRegisterLine(row, _
          linedate:=Cells(row - 1, "A").Value, _
          description:=Year(Cells(row - 1, "A").Value) & " " & _
                       MonthName(Month(Cells(row - 1, "A").Value)) & " Rent", _
          charge:="=$C$5")
    lastChargeRow = Cells(Rows.Count, "F").End(xlUp).row 'update last rent charge row
  End If
  If lastDepositRow = 12 Then
    row = writeRegisterLine(row, _
          linedate:=InputBox("Please enter the date (mm/dd/yyyy) of the initial deposit/payment
                              for the period beginning " & _
                             Cells(13, "A").Text & ".", "New Property Account Register Detected"), _
          description:="Rent Payment", _
          deposit:=InputBox("Please enter the amount of the initial deposit/payment
                              for the period beginning " & _
                            Cells(13, "A").Text & ".", "New Property Account Register Detected"))
    lastDepositRow = Cells(Rows.Count, "E").End(xlUp).row 'update last rent payment row
    If Cells(row - 1, "A").Value = "" Or Cells(row - 1, "E").Value = "" Then Exit Sub
  End If

  'If new month then insert new month's rent due
  If Month(Now) > Month(Cells(lastChargeRow, "A").Value) Then
    Do
      row = writeRegisterLine(row, _
            linedate:=DateAdd("m", 1, Cells(lastChargeRow, "A").Value), _
            description:=Year(DateAdd("m", 1, Cells(lastChargeRow, "A").Value)) & " " & _
                         MonthName(Month(DateAdd("m", 1, Cells(lastChargeRow, "A").Value))) & " Rent", _
            charge:="=$C$5")
      lastChargeRow = Cells(Rows.Count, "F").End(xlUp).row 'update last rent charge row
      DoEvents
    Loop Until Month(Now) = Month(Cells(lastChargeRow, "A").Value)
  End If

  'login to bank and download new deposit transactions
  frmLogin.Show 'show Wells Fargo login form
  If frmLogin.pressedOK = False Then Exit Sub 'the user pressed Cancel to exit the form
```

```
a.visible = True
a.openpage Cells(8, "C").Value, True
a.waitForLoad
a.document.all("userid").Value = frmLogin.txtUsername.Text
a.document.all("password").Value = frmLogin.txtPassword.Text
Unload frmLogin
a.document.all("frmSignon").submit '"submit" login credentials on Wells Fargo website
a.waitForLoad
a.position = 1
a.moveTo Right(Cells(9, "C").Value, 4)
a.moveTo "href="""
URL = a.getText("""">") 'click on deposit account link
If Not Left(URL, 53) = "https://online.wellsfargo.com/das/cgi-bin/session.cgi" Then
  MsgBox "Invalid Username or Password.", vbCritical, "Input Error"
  Exit Sub
End If
a.openpage URL, True
a.waitForLoad

'find last spreadsheet deposit on online bank transaction summary
a.position = 1
Do Until Cells(row, "A").Value = Cells(lastDepositRow, "A").Value And _
         Cells(row, "E").Value = Cells(lastDepositRow, "E").Value
  'if previous deposit not found then update page to show 6 months history
  If a.moveTo("postedHeader dateHeader"" scope=""row"">") = False Then
    a.document.all("timeFilter").Value = "4" 'value for "Last 6 Months"
    a.document.all("ddaShowForm").submit '"submit" history form
    a.waitForLoad
    a.position = 1
    a.moveTo "postedHeader dateHeader"" scope=""row"">"
  End If
  Cells(row, "A").Value = a.getText("</th>") 'get date of transaction
  a.moveTo "postedHeader depositsBusinessHeader""><span class=""OneLinkNoTx"">"
  Cells(row, "E").Value = a.getText(Chr(10)) 'get amount of transaction
Loop

'move back through transactions to capture deposits not yet recorded
Do While a.moveToPrevious("postedHeader dateHeader"" scope=""row"">")
  transDate = a.getText("</th>") 'get date of transaction
  a.moveTo "postedHeader depositsBusinessHeader""><span class=""OneLinkNoTx"">"
  transAmount = a.getText(Chr(10)) 'get amount of transaction
  If Left(transAmount, 1) = "$" Then
    row = writeRegisterLine(row, transDate, "Rent Payment", transAmount)
  End If
  DoEvents
Loop
a.openpage "https://online.wellsfargo.com/das/channel/signoff", True 'Logout
a.waitForLoad

'sort records by date to facilitate calculating late fees
ActiveSheet.Sort.SetRange Range(Cells(row - 1, "A"), "G14")
ActiveSheet.Sort.Apply

'check to see if late fees need to be charged due to delinquent rent payments
For i = 13 To row - 1
  If Cells(i, "F").Value = Cells(5, "C").Value Then 'find rent charge rows
    r = 0
    Do 'look at dates of rows after each rent charge until date > 5th of current month
      r = r + 1
      If Cells(i + r, "A").Value = "" Then Exit Do
    Loop Until Cells(i + r, "A").Value > DateAdd("d", 4, Cells(i, "A"))
    If Cells(i + r - 1, "G").Value < 0 Then 'if balance < 0 on 5th of month, then add late fee for that month
      row = writeRegisterLine(row, _
            linedate:=DateAdd("d", 5, Cells(i, "A")), _
            description:=Year(Cells(i, "A").Value) & " " & _
                         MonthName(Month(Cells(i, "A").Value)) & " Late Fee", _
            charge:="=$C$5*$C$6")
    End If
  End If
Next
```

```
  'sort records by date
  ActiveSheet.Sort.SetRange Range(Cells(row - 1, "A"), "G14")
  ActiveSheet.Sort.Apply

  'recolor lines to make up for sorting
  Range(Cells(row - 1, "A"), "G14").Interior.ThemeColor = xlThemeColorAccent5
  For i = 14 To row Step 2
    Range(Cells(i, "A"), Cells(i, "G")).Interior.TintAndShade = 0.8
    Range(Cells(i + 1, "A"), Cells(i + 1, "G")).Interior.TintAndShade = 0.6
  Next
End Sub
```

Exhibit 2: Code for *moveToPrevious* Function Within the *agent* Class

```
Function moveToPrevious(theString As String, Optional ignoreCase As Boolean) As Boolean
  Dim myPos As Long
  If ignoreCase Then
    If ucaseHTML = "" Then ucaseHTML = UCase(theHTML)
    myPos = InStrRev(ucaseHTML, UCase(theString), pos)
    myPos = InStrRev(ucaseHTML, UCase(theString), myPos)
  Else
    myPos = InStrRev(theHTML, theString, pos)
    myPos = InStrRev(theHTML, theString, myPos)
  End If

  If myPos = 0 Then
    moveToPrevious = False
  Else
    pos = myPos + Len(theString)
    moveToPrevious = True
  End If
End Function
```

Exhibit 3: Code for *writeRegisterLine* Function

```
Function writeRegisterLine(row As Long, linedate As String, description As String, Optional deposit As String, _
                           Optional charge As String, Optional balance As String) As Long
  Cells(row, "A").Value = linedate
  Cells(row, "B").Value = description
  If deposit > "" Then Cells(row, "E").Value = deposit
  If charge > "" Then Cells(row, "F").Formula = charge
  If balance > "" And row = 13 Then
    Cells(row, "G").Value = balance
  Else
    Cells(row, "G").Formula = "=R[-1]C+RC[-2]-RC[-1]"
  End If
  writeRegisterLine = row + 1
End Function
```

Exhibit 4: Code for *emailStatement* Sub Procedure

```
Sub emailStatement(Optional control As IRibbonControl) 'Callback for sendEmail onAction
  Dim templatePath As String
  Dim statementPath As String
  Dim wb As Workbook
  Dim template As Worksheet
  Dim register As Worksheet
  Dim startdate As String
  Dim templateRow As Long
  Dim registerRow As Long
  Dim userName As String
  Dim password As String
  Dim attachment As String
  Dim message As String
  Dim email As String
```

```
Dim pos As Integer
Dim subject As String

'set initial variables and worksheet objects
templatePath = ThisWorkbook.path & Application.PathSeparator & "templates" & Application.PathSeparator
statementPath = ThisWorkbook.path & Application.PathSeparator & "statements" & Application.PathSeparator
Set wb = Workbooks.Open(templatePath & "account_statement_template.xlsx")
Set template = wb.Sheets("Template")
Set register = ThisWorkbook.Sheets("Register")
registerRow = register.Cells(Rows.Count, "A").End(xlUp).row 'find row of last entry

'set account statement transaction start date (default to beginning of month previous to now)
startdate = InputBox("Please enter a transaction start date (mm/dd/yyyy) between " & _
                     register.Cells(13, "A").Value & " and " & Format(Now, "short date") & _
                     " for the emailed account statement line items.", "Account Statement Start Date", _
                     Format(DateSerial(Year(DateAdd("m", -1, Now)), _
                     Month(DateAdd("m", -1, Now)), 1), "short date"))
If startdate = "" Then Exit Sub
On Error GoTo invalidDate
Do While CLng(DateValue(startdate)) < register.Cells(13, "A").Value Or _
         CLng(DateValue(startdate)) > register.Cells(registerRow, "A").Value
  startdate = InputBox("Invalid entry!" & vbNewLine & _
                       "Please enter a valid transaction start date (mm/dd/yyyy) between " & _
                       register.Cells(13, "A").Value & " and " & Format(Now, "short date") & _
                       " for the emailed account statement line items.", "Account Statement Start Date", _
                       Format(DateSerial(Year(DateAdd("m", -1, Now)), _
                       Month(DateAdd("m", -1, Now)), 1), "short date"))
  If startdate = "" Then Exit Sub
  DoEvents
Loop
On Error GoTo 0

'insert all relevant fields into statement template
With template
  .Cells(6, "A").Value = register.Cells(3, "E").Value 'set Name
  .Cells(7, "A").Value = register.Cells(3, "C").Value 'set Address
  .Cells(8, "A").Value = register.Cells(4, "C").Value 'set City, State, Zip
  .Cells(9, "A").Value = register.Cells(4, "E").Value 'set Phone
  .Cells(10, "A").Value = register.Cells(6, "E").Value 'set email
  Do Until register.Cells(registerRow - 1, "A").Value < CLng(DateValue(startdate))
    registerRow = registerRow - 1
    If registerRow = 14 Then Exit Do
  Loop
  .Cells(13, "A").Value = register.Cells(registerRow, "A").Value 'set balance brought forward date
  .Cells(13, "B").Value = "Balance Brought Forward"
   'set balance brought forward amount
  .Cells(13, "I").Value = register.Cells(registerRow - 1, "G").Value * (-1)
  templateRow = 14
  Do Until register.Cells(registerRow, "A").Value = ""
    .Cells(templateRow, "A").Value = register.Cells(registerRow, "A").Value 'set date
    .Cells(templateRow, "B").Value = register.Cells(registerRow, "B").Value 'set description
    .Cells(templateRow, "G").Value = register.Cells(registerRow, "F").Value 'set charges
    .Cells(templateRow, "H").Value = register.Cells(registerRow, "E").Value 'set credits
    .Cells(templateRow, "I").Formula = "=R[-1]C+RC[-2]-RC[-1]" 'set balance formula
    templateRow = templateRow + 1
    registerRow = registerRow + 1
  Loop
End With

'save prepared account statement as a PDF file
template.ExportAsFixedFormat Type:=xlTypePDF, Filename:=statementPath & Year(template.Cells(3, "H").Value) & _
  " " & MonthName(Month(template.Cells(3, "H").Value)) & " Rent Statement.pdf", _
  Quality:=xlQualityStandard, IncludeDocProperties:=True, IgnorePrintAreas:=False, OpenAfterPublish:=False
attachment = statementPath & Year(template.Cells(3, "H").Value) & _
  " " & MonthName(Month(template.Cells(3, "H").Value)) & " Rent Statement.pdf"

'prepare to send account statement email
frmGmail.Show 'show Gmail login form
If frmGmail.pressedOK = False Then Exit Sub
userName = frmGmail.txtAccount.Text & "@gmail.com"
password = frmGmail.txtPassword.Text
```

```
  'read template email message into memory
  Open templatePath & "email_message_template.txt" For Input As #1
    message = Input(LOF(1), 1) 'input into message all characters (length of file-LOF) of file 1
  Close #1

  'replace coded fields in email template
  message = Replace(message, "<YR>", Year(template.Cells(3, "H").Value))
  message = Replace(message, "<MO>", MonthName(Month(template.Cells(3, "H").Value)))
  message = Replace(message, "<RENTERNAME>", Left(template.Cells(6, "A").Value, _
                    InStr(1, template.Cells(6, "A").Value, " ") - 1))
  message = Replace(message, "<PROPERTYADDRESS>", template.Cells(7, "A").Value)
  message = Replace(message, "<AGENTNAME>", frmGmail.txtAgentName.Text)
  message = Replace(message, "<AMOUNTDUE>", template.Cells(37, "A").Text)
  email = template.Cells(10, "A").Value

  'parse email subject/body
  pos = InStr(1, message, vbNewLine)
  subject = Left(message, pos - 1)
  message = Mid(message, pos + 2)

  'send email and display confirmation/failure
  If sendGMail(email, userName, password, subject, message, attachment) Then
    MsgBox "Account statement for " & MonthName(Month(template.Cells(3, "H").Value)) & " " & _
           Year(template.Cells(3, "H").Value) & " emailed to" & vbNewLine & _
           template.Cells(6, "A").Value & " (" & template.Cells(10, "A").Value & ")." , _
           vbInformation, "Email Confirmation"
  Else
    MsgBox "WARNING:" & vbNewLine & "Account statement for " & _
           MonthName(Month(template.Cells(3, "H").Value)) & " " & _
           Year(template.Cells(3, "H").Value) & " email to" & vbNewLine & _
           template.Cells(6, "A").Value & " (" & template.Cells(10, "A").Value & ") FAILED." , _
           vbCritical, "Email Failed"
  End If

  'cleanup
  wb.Close SaveChanges:=False
  Unload frmGmail
  Exit Sub

invalidDate:
  MsgBox "Invalid date entered. Exiting.", vbCritical, "Input Error"
End Sub
```

Exhibit 5: Code for *sendGMail* Function

```
Function sendGMail(sendTo As String, from As String, pw As String, subject As String, body As String, Optional
attachment As String) As Boolean
  Dim message As Object
  Dim config As Object
  Dim schema As String

  Set message = CreateObject("CDO.Message")
  Set config = CreateObject("CDO.Configuration")

  ' set up the connection to the smtp mail server
  schema = "http://schemas.microsoft.com/cdo/configuration/"
  config.Fields.Item(schema & "sendusing") = 2 ' cdoSendUsingPort
  config.Fields.Item(schema & "smtpserver") = "smtp.gmail.com"
  config.Fields.Item(schema & "smtpserverport") = 465
  config.Fields.Item(schema & "smtpauthenticate") = 1 ' cdoBasic authentication
  config.Fields.Item(schema & "sendusername") = from
  config.Fields.Item(schema & "sendpassword") = pw
  config.Fields.Item(schema & "smtpusessl") = True
  config.Fields.update

  'configure message
  message.To = sendTo
  message.from = from
```

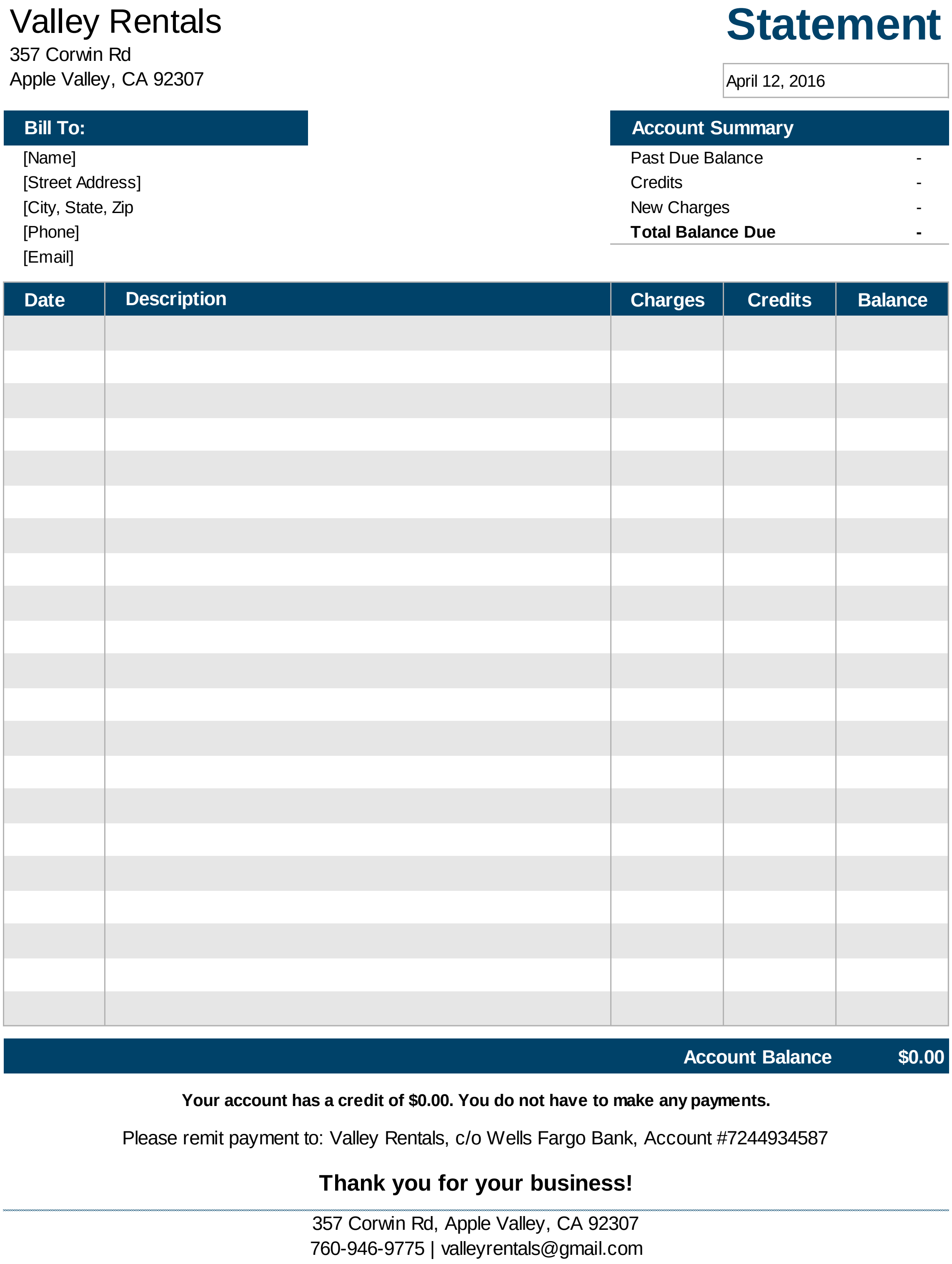
```
  message.subject = subject
  message.TextBody = body
  'message.HTMLBody = body
  'message.Sender = "Myname"
  'message.Organization = "Myname"
  'message.ReplyTo = from

  'add attachment if provided
  If attachment > "" Then
    message.AddAttachment attachment
  End If

  'connect message to configuration
  Set message.Configuration = config

  On Error Resume Next
   message.send  'send message
   If Err.Number = 0 Then
     sendGMail = True
   Else
     sendGMail = False
     Debug.Print "*****************************************************"
     Debug.Print Now
     Debug.Print "Problem sending message.  Error # " & Err.Number
     Debug.Print Err.description
     Debug.Print "To: " & sendTo
     Debug.Print "From: " & from
     Debug.Print "*****************************************************"
   End If
  On Error GoTo 0

  'cleanup
  Set message = Nothing
  Set config = Nothing
End Function
```

Exhibit 6: Code for *textStatement* Sub Procedure

```
Sub textStatement(Optional control As IRibbonControl) 'Callback for sendText onAction
  Dim userName As String
  Dim password As String
  Dim message As String
  Dim i As Integer
  Dim email As String
  Dim pos As Integer
  Dim subject As String

  'prepare to send account statement text message
  frmGmail.Show 'show Gmail login form
  If frmGmail.pressedOK = False Then Exit Sub
  userName = frmGmail.txtAccount.Text & "@gmail.com"
  password = frmGmail.txtPassword.Text

  'read template text message into memory
  Open ThisWorkbook.path & Application.PathSeparator & "templates" & Application.PathSeparator & _
       "text_message_template.txt" For Input As #1
    message = Input(LOF(1), 1) 'input into message all characters (length of file-LOF) of file 1
  Close #1

  'replace coded fields in text message template
  message = Replace(message, "<RENTERNAME>", Left(Cells(3, "E").Value, InStr(1, Cells(3, "E").Text, " ") - 1))
  message = Replace(message, "<LASTDEPAMOUNT>", Cells(Rows.Count, "E").End(xlUp).Text)
  message = Replace(message, "<LASTDEPDATE>", Cells(Cells(Rows.Count, "E").End(xlUp).row, "A").Text)
  message = Replace(message, "<RENTAMOUNT>", Cells(5, "C").Text)
  message = Replace(message, "<LATEFEE>", Format(Cells(5, "C").Value * Cells(6, "C").Value, "$#,##0.00"))
  If Cells(Rows.Count, "G").End(xlUp).Value >= 0 Then
    message = Replace(message, "<ACCOUNTBALANCE>", "Your account has a credit of " & _
                      Cells(Rows.Count, "G").End(xlUp).Text & ". You do not have to make any payments.")
  Else
```

```
    message = Replace(message, "<ACCOUNTBALANCE>", "Your account is past due! Please deposit payment of " & _
                        Mid(Cells(Rows.Count, "G").End(xlUp).Text, 2) & " to avoid eviction proceedings.")
  End If
  message = Replace(message, "<AGENTNAME>", frmGmail.txtAgentName.Text)

  'create text message email address
  For i = 1 To Len(Cells(5, "E").Value)
    If IsNumeric(Mid(Cells(5, "E").Value, i, 1)) Then email = email & Mid(Cells(5, "E").Value, i, 1)
  Next
  email = email & "@tmomail.net" 'this is currently "hard coded" for T-Mobile carrier

  'parse text message subject/body
  pos = InStr(1, message, vbNewLine)
  subject = Left(message, pos - 1)
  message = Mid(message, pos + 2)

  'send text message and display confirmation/failure
  If sendGMail(email, userName, password, subject, message) Then
    MsgBox "Account statement texted to" & vbNewLine & Cells(3, "E").Value & " (" & _
           Cells(5, "E").Value & ").", vbInformation, "Text Confirmation"
  Else
    MsgBox "WARNING:" & vbNewLine & "Account statement text to" & vbNewLine & Cells(3, "E").Value & _
           " (" & Cells(5, "E").Value & ") FAILED.", vbCritical, "Text Failed"
  End If

  'cleanup
  Unload frmGmail
End Sub
```

Exhibit 7: Account Statement Template



Valley Rentals
357 Corwin Rd
Apple Valley, CA 92307

Statement

April 12, 2016

Bill To:
[Name]
[Street Address]
[City, State, Zip
[Phone]
[Email]

Account Summary	
Past Due Balance	-
Credits	-
New Charges	-
Total Balance Due	**-**

Date	Description	Charges	Credits	Balance

Account Balance	**$0.00**

Your account has a credit of $0.00. You do not have to make any payments.

Please remit payment to: Valley Rentals, c/o Wells Fargo Bank, Account #7244934587

Thank you for your business!

357 Corwin Rd, Apple Valley, CA 92307
760-946-9775 | valleyrentals@gmail.com