The Worksheet object

introduction to the Worksheet object

This chapter will focus on the aspects of the Worksheet that appear commonly in VBA code. This chapter is a little shorter than others because in general, the Worksheet is a conduit to more useful things. There is very little that takes place within the Worksheet object that is not just a pass through to the more interesting details (e.g. Range or Chart). Having said that, there are a handful of areas that are relevant to the Worksheet and not accessible anywhere else. Those specific areas include:

- Creating and managing Worksheets – this sounds obvious but managing the references to Worksheets becomes a major issue when working with large, complicated workflows
- Print layout, printing, and exporting
- Locking and setting passwords on Worksheets
- Managing the properties of the Worksheet itself including Name, tab color, etc.

TODO: any other Worksheet things?

Of the topics listed above, the most important area is actually creating and managing the Worksheets in a complicated workflow. This is closely related to working with Ranges since presumably you create the Worksheet to put data into or something else into it. Managing the references to Worksheets can be a big deal and determining how best to access or select a given Worksheet can be important. In addition to getting references, there are a handful of times where you actually need to Activate a Worksheet. Knowing when this is and is not required is important.

TODO: when do you have to Activate?

creating and managing Worksheets

This section will focus on how to create a Worksheet and get a reference to new Worksheets. In addition to that, it will discuss managing Worksheets, including rearranging and deleting them.

references to Worksheets

The process for working with Worksheets is the same as all the other Excel Object Model objects: obtain a reference to the object and access it properties. For the Worksheet, there are a handful of ways to obtain a reference to a Worksheet. Those include:

- ActiveSheet
- Worksheets(index) or Sheets(index) global objects
- Workbook.Sheets(index) or Workbook.Worksheets(index) with a workbook object
- VBA references (Sheet1, Sheet2)
- Store the reference after creating a new sheet
- Iterating through Worksheets and picking with some criteria
- Copy a Worksheet and then search for the result (see notes below; TODO: add notes)

The basic dividing line of the methods above is when you want to access the Worksheet and what you potentially know about it. The simplest approach is when you want some code to run on the ActiveSheet because you can just ask for it. Technically, you can avoid most references to the ActiveSheet use the unqualified global references, but this can be lead to errors later. The business of obtaining a reference to a Worksheet using the other means typically only comes up when you are working with multiple Worksheets. This is quite common to do.

Once you start working with multiple Worksheets, there are a couple of common things you may want to do:

- Apply the same action to multiple sheets
- Process some data on one sheet based on the data on another sheet
- Move data from one sheet to another
- Move a chart or other object from one sheet to another
- Create a throwaway Worksheet with some information about the rest of your Workbook (e.g. output all the sheet names)

In some of those instances, you are working with multiple sheets because you want to do something (e.g. print layout or formatting) to multiple sheets. In others, you are working on multiple sheets because you know in advance that some task will use data from multiple Worksheets. For the former case, you are likely to throw you code into a loop across all Worksheets and then use some logic to determine whether or not to apply the action. In the other case, you will likely use a sheet name or index to directly access the sheet you want.

It is worth mentioning that every Workbook has built in dedicated references to the Worksheets which can be used. These exist as a part of the Object Model. By default they are called `Sheet1`, `Sheet2`, etc. These objects are always available and provide a direct reference to the worksheet. They can be quite helpful if you rename them from the default names. A couple of important items about these objects:

- They only exist as objects in the current Workbook. That is, if you want to access a Worksheet in another Workbook, this approach will not work. You can technically add a reference to the other

Workbook, but I don't recommend doing that.

- Their naming is independent of the actual sheet name displayed in Excel. This can be incredibly confusing for a new developer (especially if they are not using `Option Explicit`).
- It is very difficult to use these objects to perform some action to multiple Worksheets.

For what it's worth, I've never used the objects directly. I find myself using the sheet name directly when needed. This leads to issues with the name being changed, but at some point searching for the string in code is easier than trying to rename the object in the VBE sidebar. All of the references will break either way.

creating a Worksheet

Aside from referencing an existing Worksheet often times the core task of some automation is to create a new Worksheet. There are a number of reasons you might want to do this:

- A blank sheet is a great starting part for storing some intermediate or final result. It is nearly guaranteed to be the same every time you call for one which is much better than putting new data in an existing sheet.
- You need a blank sheet for the output of some process that is run over a number of items (each analysis gets a new sheet).
- Copying an existing Worksheet and then applying some transformation to the result.
- You created a new Workbook. This adds an extra step but leaves you with the same result as a new sheet alone (unless it was created from a template).

From my own experience, I find that creating a new Worksheet is an absolutely critical task. Very often the goal of using VBA is to automate some task over a range of inputs or possible outputs. This often means that the output for a given command may need to be produced several times. In this case, I regularly create new Worksheets instead of managing the multiple sets of data in one sheet.

In other cases, you may use a temporary intermediate new Worksheet to provide a dumping place for some calculations or other work. This is a much safer approach than to use the existing Worksheet for temporary efforts. Unless you are certain of the contents of an existing Worksheet, there is little reason to avoid creating a new one.

It's worth noting that Excel is quite performant even with a large number of Worksheets. This is especially true if the Worksheets are not linked or related via calculations. My strongest advice on this front is to liberally create new Worksheets and deal with the aftermath later. If you are building a complicated workflow, sometimes the best output is one that is useful but completely disposable. This means that the

output is impressive but due to the speed of the automation there is little reason to save or otherwise consume the resulting file. When this is the case, there is no penalty for disorganized Worksheets if the intended product is still there. Let Excel deal with the references and Ranges etc. while you deal with maintaining the references in VBA.

Having said all of that, creating a Worksheet is incredibly simple `Workbook.Sheets.Add()`. That Function will return the Worksheet object which is a reference to the new sheet. The new sheet will have a default name. THe parameters to `Add` control the location of the new sheet with respect to others. It is very, very unlikely that you will create a new Worksheet and not immediately want the sheet reference. That is, you will probably always call `Add` with a preceding `Set` to save the reference. This reference can be as good as gold in an automated workflow since an empty Worksheet is a very powerful starting (and possibly daunting) point.

If you need a copy of an existing Worksheet instead of a blank one, the command is quite simple: `Worksheet.Copy()`. This will create a Copy with parameters for location (TODO: is that true?). The major downside of using `Copy` is that it will NOT return a reference to the newly created Worksheet. This is a real travesty because it means you then have to turn around and do some work to find the newly create Worksheet. My preferred approach is to Copy the Worksheet to the first or last location in the Sheet order and then find it there. Once found, you can move the Worksheet to a desired location and then use the reference.

removing a Worksheet

If you need to delete a Worksheet, it is a simple command again: `Worksheet.Delete`. The one downside to this command is that it will fire off a warning prompt if the Worksheet contained any data or was otherwise not “blank”. This warning box will stall the execution of your VBA until it is addressed. This is a major issue for any serious workflow since your users will have to constantly click “Yes” to delete the Worksheet but they may also have no idea what they are deleting. To avoid this issue, you will nearly ALWAYS wrap the `Delete` command with the commands to disable and then re-enable the alerts. The typical code looks like:

```
Application.DisplayAlerts = False
Worksheet.Delete
Application.DisplayAlerts = True
```

When doing this dance, be absolutely certain that you re-enable the alerts. Excel will not do it for you. You may benefit from creating a new helper Sub which contains the above code as a `DeleteSheet` command

to avoid constantly adding those alerts.

TODO: add a note about when to create a new Worksheet vs. a new Workbook and the pros/cons there (maybe put this in the workflow section of book)

rearranging Worksheets

To rearrange the Worksheets, the command is simple: `Worksheet.Move(Before, After)`. The parameters there will indicate the sheet to place it before or after. The real task here is determining which sheet to reference there, but finding that reference is the same task that is described up at the top of the section.

AscendSheets.md TODO: move the AscendSheets code elsewhere or delete (not helpful here)

```
Public Sub AscendSheets()

    Application.ScreenUpdating = False
    Dim targetWorkbook As Workbook
    Set targetWorkbook = ActiveWorkbook

    Dim countOfSheets As Long
    countOfSheets = targetWorkbook.Sheets.Count

    Dim i As Long
    Dim j As Long

    With targetWorkbook
        For j = 1 To countOfSheets
            For i = 1 To countOfSheets - 1
                If UCase(.Sheets(i).name) > UCase(.Sheets(i + 1).name) Then .
                    Sheets(i).Move after:=.Sheets(i + 1)
            Next i
        Next j
    End With

    Application.ScreenUpdating = True
End Sub
```

properties and methods on the Worksheet

This section will focus on the specific properties and functions that exist for a Worksheet.

Some of the useful properties of a Worksheet include:

- Name
- Move
- Copy
- Protect/Unprotect
- Range, Cells, Rows, Columns, UsedRange
- The accessors which will give you a Collection of other objects
  - ChartObjects
  - Charts (TODO: that right?)
  - Shapes
  - ListObjects
  - PivotTables
  - Hyperlinks
  - Comments?
- TODO: add others

TODO: determine how to explain these and which to include

LockAllSheets.md

TODO: clean up this code

```
Public Sub LockAllSheets()

    Dim userPassword As Variant
    userPassword = Application.InputBox("Password to lock")

    If Not userPassword Then
        MsgBox "Cancelled."
    Else
        Application.ScreenUpdating = False

        'Changed to ActiveWorkbook so if add-in is not installed, it will
           target the active book rather than the xlam
```

```
        Dim targetSheet As Worksheet
        For Each targetSheet In ActiveWorkbook.Sheets
            On Error Resume Next
            targetSheet.Protect (userPassword)
        Next

        Application.ScreenUpdating = True
    End If

End Sub
```

OutputSheets.md

TODO: clean up this code

```
Public Sub OutputSheets()

    Dim outputSheet As Worksheet
    Set outputSheet = Worksheets.Add(Before:=Worksheets(1))
    outputSheet.Activate

    Dim outputRange As Range
    Set outputRange = outputSheet.Range("B2")

    Dim targetRow As Long
    targetRow = 0

    Dim targetSheet As Worksheet
    For Each targetSheet In Worksheets

        If targetSheet.name <> outputSheet.name Then

            targetSheet.Hyperlinks.Add _
                outputRange.Offset(targetRow), "", _
                "'" & targetSheet.name & "'!A1", , _
                targetSheet.name
            targetRow = targetRow + 1

```

```
        End If
    Next targetSheet

End Sub
```

UnlockAllSheets.md

TODO: clean up this code

```
Public Sub UnlockAllSheets()

    Dim userPassword As Variant
    userPassword = Application.InputBox("Password to unlock")

    Dim errorCount As Long
    errorCount = 0

    If Not userPassword Then
        MsgBox "Cancelled."
    Else
        Application.ScreenUpdating = False
        'Changed to ActiveWorkbook so if add-in is not installed, it will
           target the active book rather than the xlam
        Dim targetSheet As Worksheet
        For Each targetSheet In ActiveWorkbook.Sheets
            'Let's keep track of the errors to inform the user
            If Err.Number <> 0 Then errorCount = errorCount + 1
            Err.Clear
            On Error Resume Next
            targetSheet.Unprotect (userPassword)

        Next targetSheet
        If Err.Number <> 0 Then errorCount = errorCount + 1
        Application.ScreenUpdating = True
    End If
    If errorCount <> 0 Then
        MsgBox (errorCount & " sheets could not be unlocked due to bad
           password.")
```

```
28      End If
29  End Sub
```

print layout and exporting

This section will focus on the print and export related details of a Worksheet. In particular, it will focus on the details that are typically accessed through the Page Layout menu. This is one of the unique aspects of Worksheets because they are the holder of the print/export information. The main details related to this are:

- Print area
- Page layout – this is a very large object with a lot of properties to be set
- Exporting and printing

The details in this section can be a real time saver because one of the more tedious aspects of working with Excel is ensuring that your reports/graphs/data will print or export correctly. Being able to control these properties with VBA makes it possible to quickly apply the same formatting to a large number of Worksheets without having to click nine million times.

When editing the Page Layout, you can change nearly everything. The one thing to be aware of is related to printers. There are a number of settings in the Worksheet that are internally tied to the default (or active) printer. This shows up if you are attempting to set the page size specifically. IF you always use the same printer or have coworkers who use the same printers, you amy not notice these issues. It becomes a serious problem when you ar etrying to make code work for multiple different printers that support or identify page sizes differently.

The best way to see what is savialable for page settings is to record a macro and change one thing. Excel is a bit aggressive at including all possible settings that could have changed. This is very nice if you want to grab some settings nad work them into your code.

There are a couple of other items to describe so you know what they are:

- Using `Zoom` and `FitToPages` to set the number of pages that the output will be included in (TODO: review)
- TODO: add others

Also, be aware that changing the print settings is a per Worksheet change. This amy be obvious since the prepares are off the Worksheet, but it is easy to forget this. The nice thing however s that you can just

iterate your Worksheets and apply the same settings to all of them. This is one of the greatest time savers compared to changing properties in Excel (TODO: can these be changed with multi selection?).

Rand_common print settings

TODO: clean up this code

```
Sub Rand_CommonPrintSettings()

    Application.ScreenUpdating = False
    Dim sht As Worksheet

    For Each sht In Sheets
        sht.PageSetup.PrintArea = ""
        sht.ResetAllPageBreaks
        sht.PageSetup.PrintArea = ""

        With sht.PageSetup
            .LeftHeader = ""
            .CenterHeader = ""
            .RightHeader = ""
            .LeftFooter = ""
            .CenterFooter = ""
            .RightFooter = ""
            .LeftMargin = Application.InchesToPoints(0.75)
            .RightMargin = Application.InchesToPoints(0.75)
            .TopMargin = Application.InchesToPoints(1)
            .BottomMargin = Application.InchesToPoints(1)
            .HeaderMargin = Application.InchesToPoints(0.5)
            .FooterMargin = Application.InchesToPoints(0.5)
            .PrintHeadings = False
            .PrintGridlines = False
            .PrintComments = xlPrintNoComments
            .PrintQuality = 600
            .CenterHorizontally = False
            .CenterVertically = False
            .Orientation = xlLandscape
            .Draft = False
            .PaperSize = xlPaperLetter
```

```
            .FirstPageNumber = xlAutomatic
            .Order = xlDownThenOver
            .BlackAndWhite = False
            .Zoom = False
            .FitToPagesWide = 1
            .FitToPagesTall = False
            .PrintErrors = xlPrintErrorsDisplayed
            .OddAndEvenPagesHeaderFooter = False
            .DifferentFirstPageHeaderFooter = False
            .ScaleWithDocHeaderFooter = True
            .AlignMarginsHeaderFooter = False
            .EvenPage.LeftHeader.Text = ""
            .EvenPage.CenterHeader.Text = ""
            .EvenPage.RightHeader.Text = ""
            .EvenPage.LeftFooter.Text = ""
            .EvenPage.CenterFooter.Text = ""
            .EvenPage.RightFooter.Text = ""
            .FirstPage.LeftHeader.Text = ""
            .FirstPage.CenterHeader.Text = ""
            .FirstPage.RightHeader.Text = ""
            .FirstPage.LeftFooter.Text = ""
            .FirstPage.CenterFooter.Text = ""
            .FirstPage.RightFooter.Text = ""
            .PrintTitleRows = ""
            .PrintTitleColumns = ""
        End With
    Next sht

    Application.ScreenUpdating = True
End Sub
```