overview of charting

Charting will be a major chapter in this book since it's the focus of a lot of what I do. It also is a spot where a significant amount of time can be saved since the chart options are awful to mess with.

Some high level topics:

- Creating a chart
- Formatting a chart
- Manipulating the series on a chart
- Changing the layout of charts on a page (make grid code)
- Common patterns when working through charts (`ForEach` loops wherever possible)

introduction to charting

Charting is the second most important aspect of automatic Excel behind manipulating `Ranges`. There is a bias when saying that because a lot of what I do after engineering calculations is chart the results. In particular, Excel can be used to great effect to chart time series of data. The other reason charts are so amenable to VBA is that very often you are applying the same actions to the charts. In that sense, the VBA related to charts is doing a lot of changing settings and formats so that the charts look the way you want. This ahs the immediate effect of making your charts look less like "they came from Excel" which is a common knock in some circles.

When working with `Charts`, there is a `Range` of difficulties depending on what you are trying to do. In some cases, working with an existing `chart` is much easier than creating a new one. In other instances, it can be much simpler to create a new chart, starting from a default, rather than change all the settings back. One other major difference between `Charts` and `Ranges` is that working with charts is much more about knowing the object model than knowing how to program. The vast majority of your code related to charts is simply iterating through objects to find the one property that you want to change. This makes it easier to write chart VBA once you have the basics of `For Each` loops down. It also means that you need to spend some time getting comfortable with the object model.

There is one oddity related to Charts that is worth mentioning now. Charts can either be embedded as an object on a `Worksheet`, or they can be their own `Sheets`. I personally never use the latter case, but it is common enough that it needs to be on your mind when working with Charting code.

(I don't use the Chart as a Sheet model because I find that it is not necessary in terms of displaying data. In particular, you are at the mercy of your window size and cannot easily change the dimensions. Also, it complicates the VBA side of things to work in both formats all the time, so I just decided to always put my

Charts on Sheets. Your mileage may vary so I'll touch on both approaches in the code samples.)

a quick overview of the object model

- `ChartObjects` -> `ChartObject` - this derives from `Shape` and exists when the Chart is on a Worksheet
  - `Chart`
    - `SeriesCollection` -> `Series`
    - `Axes` -> `Axis`
    - `ChartArea`
    - `PlotArea`
- `ActiveChart` -> `Chart` - this works whether you have a Worksheet or Chart on a sheet
- `Selection` -> `Variant` - this one can be useful but is often not of the type that you want.

obtaining a reference to a Chart

When working with `Charts`, the first task is typically to get a reference to an existing chart – unless you are creating a new chart. To obtain a reference to a chart, there are a handful of ways of doing it depending on what your spreadsheet contains and how it's structured.

THe main ways to do it are:

- Use the `ActiveChart` object
- Use the `Selection` object – this is highly depending on what is selected
- Use the `ChartObjects` object
  - If you know which chart you want, you can supply an index; this works great if there is only a single chart - `ChartObjects(1)`
  - If you want to do something to all charts, you can iterate this object
  - If you have named the chart (more on that later) you can supply the name as the index - `ChartObjects("SomeChart")`
- The `Workbook.Sheets` object if your charts are contained in their own sheets
  - Same as above, you can access via a numeric index, name, or iterate through all of them

`ActiveChart` `ActiveChart` is similar to the other `Active` objects in that it does about what you expect. The one difference is that the Chart actually has to be selected or have focus in order to be considered "active". This is similar but also different to something like `ActiveWorkbook` where having the workbook open makes it active.

Note that `ActiveChart` will work for a `Chart` that is contained on a Worksheet or also for one that is its own Sheet. If the latter case, then `ActiveSheet` and `ActiveChart` will refer to the same object. Side note: this technicality is why you will not get proper Intellisense when using `ActiveSheet` – that Sheet could technically be a Chart.

The nice thing about `ActiveChart` is that it gives you the Chart object which then gives you immediate access to the Chart related details you are like to want to change. The downside is that unless you have a single Chart that is already selected, `ActiveChart` has limited application when using VBA. Again, the goal is to avoid selecting objects in order to access them via VBA so `ActiveChart` is not ideal.

`Selection` The Selection object is probably the greatest catch all for an object. It literally holds anything, and this means that using the object requires knowing what is selected, or checking vigorously before using the object. Technically, you also let your code error out if the wrong object is selected, and this works well at times. This works well because you are unlikely to be using `Selection` in a complicated workflow because, again, you should not be selecting objects to access them. This means that `Selection` is really limited to one-off and helper code where you can more tightly dictate that this code only works if you select a Chart. You should still add some error handling, but sometimes that step is skipped.

Since the `Selection` can hold anything, it's important to know what could be Selected. Related to charts, the following can all live in the `Selection`:

- `ChartObjects`
- `Chart`
- `ChartArea`
- `PlotArea`
- `Legend`
- `ChartTitle`
- `Series`

If you are writing VBA to work on Charts, you can technically require the user to select the correct part of the chart and always use `Selection`. You will quickly grow tired of having to remember which part of the Chart to select in order to make the code work. To avoid this scenario, it is helpful to remember the object model and know how to work your way around a Chart.

My approach has always been to convert the `Selection` to a Collection of `ChartObjects`. I can then always iterate that resulting Collection to process the Charts. If only a single Chart was selected, the code works all the same. The downside to this approach is that a Chart as a Sheet cannot live inside a ChartObject. This is a large part of why I always put Charts on a Worksheet.

Below is the helper function I use in order to convert a possibly Chart containing selection into a Collection of `ChartObjects`. It works for all objects except for the Axis related ones.

TODO: consider improving this code if it is included as a de facto reference

```
Public Function Chart_GetObjectsFromObject(ByVal inputObject As Object) As
    Variant

    Dim chartObjectCollection As New Collection

    'NOTE that this function does not work well with Axis objects.  Excel
        does not return the correct Parent for them.

    Dim targetObject As Variant
    Dim inputObjectType As String
    inputObjectType = TypeName(inputObject)

    Select Case inputObjectType

        Case "DrawingObjects"
            'this means that multiple charts are selected
            For Each targetObject In inputObject
                If TypeName(targetObject) = "ChartObject" Then
                    'add it to the set
                    chartObjectCollection.Add targetObject
                End If
            Next targetObject

        Case "Worksheet"
            For Each targetObject In inputObject.ChartObjects
                chartObjectCollection.Add targetObject
            Next targetObject

        Case "Chart"
            chartObjectCollection.Add inputObject.Parent

        Case "ChartArea", "PlotArea", "Legend", "ChartTitle"
            'parent is the chart, parent of that is the chart targetObject
            chartObjectCollection.Add inputObject.Parent.Parent

        Case "Series"
            'need to go up three levels
            chartObjectCollection.Add inputObject.Parent.Parent.Parent

        Case "Axis", "Gridlines", "AxisTitle"
            'these are the oddly unsupported objects
            MsgBox "Axis/gridline selection not supported.  This is an Excel
                bug.  Select another element on the chart(s)."

        Case Else
            MsgBox "Select a part of the chart(s), except an axis."

    End Select

    Set Chart_GetObjectsFromObject = chartObjectCollection
End Function
```

ChartObjects If you are working on a Worksheet, then that Worksheet will have the `ChartObjects` object. This object is great because it contains all of the Charts in their own collection (separate from any other Shapes or buttons). This `ChartObjects` collection contains object of type ChartObject. The ChartObject derives from Shape which means it contains all of the properties related to on-sheet position and size.

A typical workflow is included below since it is a pattern that shows up all the time in VBA code related to charts. At a high level the steps are:

- Use ActiveSheet or a Worksheet reference to access the `ChartObjects`
- Iterate through each `ChartObject`, storing a reference to the underlying Chart
- You then setup sections to work through the parts of the Chart you want
  - Iterate through the `SeriesCollection`
  - Iterate through the Axes
  - Touch the other top level properties including `ChartTile`, `Legend`, etc.

This workflow is quite powerful because it can quickly be wrapped with a loop to go through all Worksheets and even possible all Workbooks. It's also powerful because you can be quite comfortable learning this pattern and then adding in the parts that you actually want to change. The only downside is that it can be quite tedious to type out all the loops every time, but there's not a good way around that other than to use the clipboard.

Another approach to using `ChartObjects` is to not iterate through all of them but instead to select a single ChartObject and work with it. There are two ways to do this:

- Use an integer index for the Chart – this is quite easy to do if there are only a few charts
- Name the chart and use that name

When using either of these approaches, it is quite helpful to show the `Selection Pane` window in Excel. This pane will pop out and tell you the order and the names of all the objects on the sheet (this includes comments, shapes, and Charts). From this pane, you can rearrange the charts into the order you want or rename them.

Although `For Each` loops are generally preferred when working with Charts, sometimes you simply know that you want to change one chart and an index just lets you do that. If you are in the habit of using loops however, you can easily do that with the helper code included above which stick a single chart into a Collection.

Workbook.Sheets to get Chart references The final approach to obtaining a Chart reference is to use the `Sheets` object. Aside from ActiveChart, this is the only way to deal with Charts that are their own

Sheet. Again, you can either use an index or a Name. Here, the Name is easily changed on the Sheet tab so it's much more common to use a Name when doing this. The other approach is to iterate through all the `Sheets` and pick off the ones that are Charts.

There are two key points when working with Charts as `Sheets`:

- You must use the `Workbook.Sheets` object to access them and not `Workbook.Worksheets`. The latter object contains only those `Worksheets` that are not Charts. The former contains both Charts and `Worksheets`.
- It's possible that your Sheet is not actually a Chart. You should check the type of the object if you are going to iterate through all `Worksheets`. Also be aware that some sheets can be hidden which might lead to unexpected results.

TODO: is there a Charts object on Workbook?

common objects/properties for a Chart

This section will focus on the common formatting changes that can be made to a Chart. The next section focuses on creating a Chart from scratch if you want to see that. These common changes will be grouped by the type that they affect, but this is not meant to be an exhaustive list. Instead, this is a list that will cover the objects nad functions that are actually used in regular code. There will be several other things that you will need to check the reference for (or record a macro), but this listing will get you started with the regular things.

To organize this section, we will focus on the different parts of a Chart in turn along with how to access the things you need. This section is meant to be a one stop shop for working on the common parts of a Chart. This will cover:

- `ChartObject`
  - Top, Left, Height, Width - control the location of a chart
- `Chart`
  - ChartType
  - Access the other objects and controls whether some things exist
    * HasLegend
    * HasTitle
- `Legend`
- `Series` – accessed the the Chart.SeriesCollection
  - ChartType
- `Axis` – accessed through Chart.Axes

    - Display the axis
    - Change the text
    - Change the min/max scale including automatic values
    - Change the number format of the axis
    - Change the format and display of the Gridlines
- `Point` – accessed through a Series
    - Change display of individual points
    - Control the DataLabels (HasLabel and then DataLabel)
- `Trendline`

TODO: go through bUTL and find other commonly appearing things

common changes to the ChartObject

The ChartObject is the main container for a Chart that is on a Worksheet. The common changes then are related to the position and size of the Chart on the Worksheet. The common properties to change here are:

- Top
- Left
- Height

- Width
- Placement (controls the move with cells option)

All of these are of type Double which means you can use decimal calculations to determine the size or position. In Excel, the 0,0 point is at the upper left hand corner (upper left of cell A1) and the Top and Left increase going to the right and down. If you are familiar with 0,0 being the center of the XY plane, then Excel will be a tad unfamiliar. Once you get used to it, you will realize that there is not really a better way to arrange the coordinate system since the spreadsheet can extend to the right and down nearly infinitely.

TODO: are there Bottom and Right properties too?

TODO: add a comment about Points vs. inches here and the function to convert them

The most common application of changing these properties is to either standardize the size of several charts or to arrange the charts in a grid (which standardizes the size and then position).

That code is included below:

TODO: clean up this code to only the required parts

```
Public Sub Chart_GridOfCharts( _
    Optional columnCount As Long = 3, _
    Optional chartWidth As Double = 400, _
    Optional chartHeight As Double = 300, _
    Optional offsetVertical As Double = 80, _
    Optional offsetHorizontal As Double = 40, _
    Optional shouldFillDownFirst As Boolean = False, _
    Optional shouldZoomOnGrid As Boolean = False)

    Dim targetObject As ChartObject

    Dim targetSheet As Worksheet
    Set targetSheet = ActiveSheet

    Application.ScreenUpdating = False

    Dim countOfCharts As Long
    countOfCharts = 0

    For Each targetObject In targetSheet.ChartObjects
        Dim left As Double, top As Double

        If shouldFillDownFirst Then
            left = (countOfCharts \ columnCount) * chartWidth +
                offsetHorizontal
            top = (countOfCharts Mod columnCount) * chartHeight +
                offsetVertical
        Else
            left = (countOfCharts Mod columnCount) * chartWidth +
                offsetHorizontal
            top = (countOfCharts \ columnCount) * chartHeight +
                offsetVertical
        End If

        targetObject.top = top
        targetObject.left = left
        targetObject.Width = chartWidth
        targetObject.Height = chartHeight

        countOfCharts = countOfCharts + 1

```

```
    Next targetObject

    'loop through columns to find how far to zoom
    'Cells.Left property returns a variant in points
    If shouldZoomOnGrid Then
        Dim columnToZoomTo As Long
        columnToZoomTo = 1
        Do While targetSheet.Cells(1, columnToZoomTo).left < columnCount *
            chartWidth
            columnToZoomTo = columnToZoomTo + 1
        Loop

        targetSheet.Range("A:A", targetSheet.Cells(1, columnToZoomTo - 1).
            EntireColumn).Select
        ActiveWindow.Zoom = True
        targetSheet.Range("A1").Select
    End If

    Application.ScreenUpdating = True
```

```

End Sub
```

common properties of the Chart

The Chart object is mostly a container for the other more useful properties of the Chart, but there are a couple of common properties that live at this top level. Those include:

- The HasXXX: HasTitle, HasLegend (TODO: any others?) - control the display of these things
- ChartType
- Delete
- Copy (TODO: this on ChartObject also?)

TODO: find more of these

In addition to those properties, the Chart object provides access to other useful things via the common accessors:

- SeriesCollection
- Axes
- Legend

- ChartTitle
- ChartArea
- PlotArea

TODO: is this list complete?

common properties of the Series

One of the two most used Chart objects is the Series (other is the Axis). The Series ends up being powerful because it provides access to the data of the Chart along with the major formatting choices since the Series is the prominent feature of a Chart.

The common things to go after for a series are:

- Data related
  - Name
  - XValues
  - Values
  - Formula
- Formatting related

    - Format
        * Line
    - MarkerSize
    - MarkerStyle
    - MarkerForegroundColor, MarkerBackgroundColor

Also, from a Series you can access the following other objects:

- Points
- Trendlines

common properties of the Axis

The Axis is the second most common object to work with (behind the Series). This is largely because the Axis controls or provides access to a lot of the formatting related aspects of the Chart. The Axis also controls the scale of the Axis and in that regard, is a critical part of making or editing a Chart.

The first part of the Axis is accessing the correct one. This is slightly tricky the first time because the Axes are stored in the Chart.Axes object. THe real trick is that this object is indexed by the xlAxisType (TODO: check that) which can be xlCategory (for the x-axis) or xlValue/xlValue2 (for the y-axis, left and right).

Once you have an Axis object, you can set to work changing the common properties:

- Scale related
  - MinimumScale/MaximumScale
  - MinimumScaleIsAuto/MaximumScaleIsAuto
- Formatting related (most of these are accessors to a different object)
  - GridLines (Major/minor and the HasXXX)
  - Ticks (TODO: that right?)
  - HasTitle and AxisTitle

Chart_Axis_AutoX.md

```
Public Sub Chart_Axis_AutoX()

    Dim targetObject As ChartObject
    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        Dim targetChart As Chart
        Set targetChart = targetObject.Chart

        Dim xAxis As Axis
        Set xAxis = targetChart.Axes(xlCategory)
        xAxis.MaximumScaleIsAuto = True
        xAxis.MinimumScaleIsAuto = True
        xAxis.MajorUnitIsAuto = True
        xAxis.MinorUnitIsAuto = True

    Next targetObject

End Sub
```

Chart_Axis_AutoY.md

```
Public Sub Chart_Axis_AutoY()

    Dim targetObject As ChartObject
    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        Dim targetChart As Chart
        Set targetChart = targetObject.Chart

        Dim yAxis As Axis
        Set yAxis = targetChart.Axes(xlValue)
        yAxis.MaximumScaleIsAuto = True

        yAxis.MinimumScaleIsAuto = True

        yAxis.MajorUnitIsAuto = True

        yAxis.MinorUnitIsAuto = True


    Next targetObject


End Sub
```

Chart_AxisTitleIsSeriesTitle.md

```
Public Sub Chart_AxisTitleIsSeriesTitle()


    Dim targetObject As ChartObject

    Dim targetChart As Chart

    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        Set targetChart = targetObject.Chart


        Dim butlSeries As bUTLChartSeries

        Dim targetSeries As series

        For Each targetSeries In targetChart.SeriesCollection
            Set butlSeries = New bUTLChartSeries
            butlSeries.UpdateFromChartSeries targetSeries

            targetChart.Axes(xlValue, targetSeries.AxisGroup).HasTitle = True
            targetChart.Axes(xlValue, targetSeries.AxisGroup).AxisTitle.Text = butlSeries.name

            '2015 11 11, adds the x-title assuming that the name is one cell above the data
            '2015 12 14, add a check to ensure that the XValue exists
            If Not butlSeries.XValues Is Nothing Then
                targetChart.Axes(xlCategory).HasTitle = True
                targetChart.Axes(xlCategory).AxisTitle.Text = butlSeries.XValues.Cells(1, 1).Offset(-1).Value
            End If

        Next targetSeries
```

```
    Next targetObject
End Sub
```

Chart_FitAxisToMaxAndMin.md

```
Public Sub Chart_FitAxisToMaxAndMin(ByVal axisType As XlAxisType)

    Dim targetObject As ChartObject
    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        '2015 11 09 moved first inside loop so that it works for multiple
            charts
        Dim isFirst As Boolean
        isFirst = True

        Dim targetChart As Chart
        Set targetChart = targetObject.Chart

        Dim targetSeries As series
        For Each targetSeries In targetChart.SeriesCollection

```

```
            Dim minSeriesValue As Double
            Dim maxSeriesValue As Double

            If axisType = xlCategory Then

                minSeriesValue = Application.Min(targetSeries.XValues)
                maxSeriesValue = Application.Max(targetSeries.XValues)

            ElseIf axisType = xlValue Then

                minSeriesValue = Application.Min(targetSeries.Values)
                maxSeriesValue = Application.Max(targetSeries.Values)

            End If

            Dim targetAxis As Axis
            Set targetAxis = targetChart.Axes(axisType)

            Dim isNewMax As Boolean, isNewMin As Boolean
            isNewMax = maxSeriesValue > targetAxis.MaximumScale
            isNewMin = minSeriesValue < targetAxis.MinimumScale

            If isFirst Or isNewMin Then targetAxis.MinimumScale =
               minSeriesValue
            If isFirst Or isNewMax Then targetAxis.MaximumScale =
               maxSeriesValue

            isFirst = False
        Next targetSeries
    Next targetObject

End Sub
```

Chart_YAxisRangeWithAvgAndStdev.md

```
Public Sub Chart_YAxisRangeWithAvgAndStdev()

    Dim numberOfStdDevs As Double

    numberOfStdDevs = CDbl(InputBox("How many standard deviations to include? "))

    Dim targetObject As ChartObject

    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        Dim targetSeries As series
        Set targetSeries = targetObject.Chart.SeriesCollection(1)

        Dim avgSeriesValue As Double
        Dim stdSeriesValue As Double

        avgSeriesValue = WorksheetFunction.Average(targetSeries.Values)
        stdSeriesValue = WorksheetFunction.StDev(targetSeries.Values)

        targetObject.Chart.Axes(xlValue).MinimumScale = avgSeriesValue - stdSeriesValue * numberOfStdDevs
        targetObject.Chart.Axes(xlValue).MaximumScale = avgSeriesValue +
            stdSeriesValue * numberOfStdDevs

    Next

End Sub
```

common properties of the Legend

The Legend is a simple affair compared to the others. There really only two things to do with it: remove it or move it. Both of these are simple enough:

- HasLegend (on the Chart)
- Delete
- Position

TODO: add an example of these in action

common properties of a Point

The Point represents the lowest level when it comes to how the data and formatting of a Chart is built. In general, you do not have to actively go editing Points. This is because you will typically edit the appearance of the Series and the Axes to get the Chart that you want. There are however times when you get down to the metal and edit the properties of the individual points. Before describing how to do this, it may help to give an example or two for why you want to get down to this level:

- Delete a data point without touching the Series
- Add a DataLabel to the point if the value is below some threshold (or if some other Range has a value)
- Hide a Point from one series because you want it to show up in another one

Of the tasks above, only one of them (the second) has to be accomplished via the Points. The others *could* be done via a different method, but you might find yourself in a spot where iterating some Points will save a ton of headache elsewhere. A cautionary note is that typically you should not be editing the properties of a Point; there is nearly always a better way to do these things. Part of the problem is that the settings you change will be quickly overwritten by changes in Excel or VBA. If you know you just need something done however, Points can be a quick way to make it happen.

TODO: look into ErrorBars here?

WHen thinking about working through the Points of a Series, consider the common properties you can change:

- HasLabel / DataLabel
- Value
- Formatting? (TODO: what are these)
- Hidden

TODO: finish this list

Note that in addition to the common properties, you can also change anything that can be changed from the normal Excel settings/properties window.

common properties of the TrendLine

The TrendLine is one of the lesser used properties, but it can be a real time saver when using VBA if you need to. The problem with the trendline normally is that you are required to work through a ton of menus to configure the properties. This is even more painful when you've got to do the same thing to multiple Series in a Chart or across multiple Charts. Similar to the other objects here, you can use VBA to quickly do the task that is otherwise a pain.

The most likely properties you'll use:

- Creating one off of a series
- Type
- Parameter

TODO: confirm these are correct

TODO: add an example showing how to add a Trendline for every Series

Chart_AddTrendlineToSeriesAndColor.md

```
Public Sub Chart_AddTrendlineToSeriesAndColor()

    Dim targetObject As ChartObject

    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        Dim chartIndex As Long
        chartIndex = 1

        Dim targetSeries As series
        For Each targetSeries In targetObject.Chart.SeriesCollection

```

```
            Dim butlSeries As New bUTLChartSeries
            butlSeries.UpdateFromChartSeries targetSeries

            'clear out old ones
            Dim j As Long
            For j = 1 To targetSeries.Trendlines.Count
                targetSeries.Trendlines(j).Delete
            Next j

            targetSeries.MarkerBackgroundColor = Chart_GetColor(chartIndex)

            Dim newTrendline As Trendline
            Set newTrendline = targetSeries.Trendlines.Add()
            newTrendline.Type = xlLinear
            newTrendline.Border.Color = targetSeries.MarkerBackgroundColor

            '2015 11 06 test to avoid error without name
            '2015 12 07 dealing with multi-cell Names
            'TODO: handle if the name is not a range also
            If Not butlSeries.name Is Nothing Then
                newTrendline.name = butlSeries.name.Cells(1, 1).Value
            End If

            newTrendline.DisplayEquation = True
            newTrendline.DisplayRSquared = True
            newTrendline.DataLabel.Format.TextFrame2.TextRange.Font.Fill.ForeColor.RGB = Chart_GetColor(chartIndex)

            chartIndex = chartIndex + 1
        Next targetSeries

    Next targetObject
End Sub
```

creating charts from scratch

The previous section discussed how to work with existing Charts. This section will focus on how to create those Charts from scratch if you are coming into a blank Worksheet or if you simply need to add a chart to existing data. At the start, it's worth mentioning that creating Charts from scratch falls into one of two categories:

- Library/helper type code where you want to quickly create a Chart in a common way. This type of code works best in an addin and typically provides functionality that you wish Excel had from the start
- One-off code for a specific application. This involves creating a Chart with some sort of odd manipulation or formatting or other detail where automation saves time.

The two types of category will end up with code that looks similar, but the goals of the former category will be slightly different than the latter. Typically when making code for a one-off application, you can make more assumptions about how the data is structured and what sorts of actions need to be taken. When working with helper code, you will spend more time asking for user input, and handling the different cases that might come up.

Another key point to make is that the type of work that is being done in a chart can vary as well. The splitting line here is whether the Chart creation is data heavy or formatting heavy (or possibly both). For a data heavy Chart, you will spend a lot of time collecting Ranges, creating Series, and possibly manipulating individual Points. For a formatting heavy chart, you will spend a lot of time iterating through the Series to apply formatting, label the Axes, set the number formats, and generally modify the Excel defaults. Both of these tasks are very time intensive if you are doing them without VBA, so both lend themselves to being automated if possible.

Excel provides two means of creating a Chart depending on how you want to handle things. Those two commands are:

- ChartObjects.Add
- TODO: what is the other method

I always prefer to use ChartObjects.Add because of it consistent application. The other approach tends to put you at the mercy of how Excel interprets your data and its layout.

TODO: add more detail here

The general process for creating a chart looks like this:

- Create a new ChartObject via ChartObjects.Add - store that reference
  - If you know where you want the Chart to go, you can use that information here
- Access the Chart of that object
- Change the properties of the Chart that you know – namely ChartType
- Access the SeriesCollection of the Chart and call NewSeries for each Series you want - store a reference

to that Series

  - This is typically done inside a loop that is iterating through Ranges in some way
  - If you need to apply Series specific formatting, do that here

- Access the Axes collection and modify any specific parts of the Axes that you want
  - This may show up in the loop above if you want the Axis to draw information from the Series (maybe set the max to the max of the data?)

At this point, you will have a Chart with the Series you want along with the major formatting taken care of. Even better, this general framework lends itself nicely to adding new commands where needed. If you need to go after some of the finer details of the Chart, you can add those commands where the objects are being reference, or at the end of the code. The main thing to consider is whether you need to work inside loops (per Series) or if you can process the extra stuff at the end.

The other upside of this approach is that you can quickly wrap all of this code with another loop to create multiple Charts. You can then wrap that code with another loop to do it on multiple Worksheets, etc. When you write code that can cleanly live inside a loop, you make it easy to use the code elsewhere.

One other aspect of Charts that is somewhat unique is that you can typically reuse a lot of the code by creating new Subs. These can be called from the inside of a loop to create a chain of commands to process a Chart. This approach is highly effective if you work in an environment where the same or similar things need to be done. For example: you have a monthly report to create each month for multiple departments. Standardizing as much of that work into modules makes it easy to apply the code in multiple spots with minor changes. This is relevant to Charts because most of the work of Charts is changing the values of specific properties. There is typically far less logic that is unique to an application (like trying to build a Range based on the layout of data).

Once you have this general framework mastered, you can quickly use it to make more charts.

TODO: add some examples of creating Charts

specific charting examples

This section will focus on some specific applications of applying VBA to charts. The code here can be quickly reused for your own application. These examples include:

- Creating a grid of XY scatter plots (a scatter matrix) based on a block of data
- Creating a panel of time series, one chart per each value with a common x-axis

TODO: identify the examples to include here

creating an XY scatter matrix

ChartCreateXYGrid.md

```
Public Sub ChartCreateXYGrid()

    On Error GoTo ChartCreateXYGrid_Error

    DeleteAllCharts
    'VBA doesn't allow a constant to be defined using a function (rgb) so we
       use a local variable rather than
    'muddying it up with the calculated value of the rgb function
    Dim majorGridlineColor As Long
    majorGridlineColor = RGB(200, 200, 200)
    Dim minorGridlineColor As Long
    minorGridlineColor = RGB(220, 220, 220)

    Const CHART_HEIGHT As Long = 300
    Const CHART_WIDTH As Long = 400
    Const MARKER_SIZE As Long = 3
    'dataRange will contain the block of data with titles included
    Dim dataRange As Range
    Set dataRange = Application.InputBox("Select data with titles", Type:=8)

    Application.ScreenUpdating = False

    Dim rowIndex As Long, columnIndex As Long
    rowIndex = 0

    Dim xAxisDataRange As Range, yAxisDataRange As Range
    For Each yAxisDataRange In dataRange.Columns
        columnIndex = 0

        For Each xAxisDataRange In dataRange.Columns
            If rowIndex <> columnIndex Then
                Dim targetChart As Chart
                Set targetChart = ActiveSheet.ChartObjects.Add(columnIndex *
                    CHART_WIDTH, _
                                                               rowIndex *
                                                                   CHART_HEIGHT
                                                                    + 100, _
```

```
                                                                  CHART_WIDTH,
                                                                     CHART_HEIGHT
                                                                     ).Chart

                 Dim targetSeries As series
                 Dim butlSeries As New bUTLChartSeries

                 'offset allows for the title to be excluded
                 Set butlSeries.XValues = Intersect(xAxisDataRange,
                    xAxisDataRange.Offset(1))
                 Set butlSeries.Values = Intersect(yAxisDataRange,
                    yAxisDataRange.Offset(1))
                 Set butlSeries.name = yAxisDataRange.Cells(1)
                 butlSeries.ChartType = xlXYScatter

                 Set targetSeries = butlSeries.AddSeriesToChart(targetChart)

                 targetSeries.MarkerSize = MARKER_SIZE
                 targetSeries.MarkerStyle = xlMarkerStyleCircle
```

```

        Dim targetAxis As Axis
        Set targetAxis = targetChart.Axes(xlCategory)
        targetAxis.HasTitle = True
        targetAxis.AxisTitle.Text = xAxisDataRange.Cells(1)
        targetAxis.MajorGridlines.Border.Color = majorGridlineColor
        targetAxis.MinorGridlines.Border.Color = minorGridlineColor

        Set targetAxis = targetChart.Axes(xlValue)
        targetAxis.HasTitle = True
        targetAxis.AxisTitle.Text = yAxisDataRange.Cells(1)
        targetAxis.MajorGridlines.Border.Color = majorGridlineColor
        targetAxis.MinorGridlines.Border.Color = minorGridlineColor

        targetChart.HasTitle = True
        targetChart.ChartTitle.Text = yAxisDataRange.Cells(1) & " vs.
            " & xAxisDataRange.Cells(1)
        'targetChart.ChartTitle.Characters.Font.Size = 8
        targetChart.Legend.Delete
```

```
            End If

            columnIndex = columnIndex + 1
        Next xAxisDataRange

        rowIndex = rowIndex + 1
    Next yAxisDataRange

    Application.ScreenUpdating = True

    dataRange.Cells(1, 1).Activate

    On Error GoTo 0
    Exit Sub

ChartCreateXYGrid_Error:

    MsgBox "Error " & Err.Number & " (" & Err.Description & _
           ") in procedure ChartCreateXYGrid of Module Chart_Format"
    MsgBox "This is most likely due to Range issues"

End Sub
```

creating a panel of time series plots

Chart_TimeSeries.md

```
Public Sub Chart_TimeSeries(ByVal rangeOfDates As Range, ByVal dataRange As
   Range, ByVal rangeOfTitles As Range)

    Application.ScreenUpdating = False
    Const MARKER_SIZE As Long = 3
    Dim majorGridlineColor As Long
    majorGridlineColor = RGB(200, 200, 200)
    Dim chartIndex As Long
    chartIndex = 1

    Dim titleRange As Range
    Dim targetColumn As Range

    For Each titleRange In rangeOfTitles

        Dim targetObject As ChartObject
        Set targetObject = ActiveSheet.ChartObjects.Add(chartIndex * 300, 0,
            300, 300)

        Dim targetChart As Chart
        Set targetChart = targetObject.Chart
        targetChart.ChartType = xlXYScatterLines
        targetChart.HasTitle = True
        targetChart.Legend.Delete

        Dim targetAxis As Axis
        Set targetAxis = targetChart.Axes(xlValue)
        targetAxis.MajorGridlines.Border.Color = majorGridlineColor

        Dim targetSeries As series
        Dim butlSeries As New bUTLChartSeries

        Set butlSeries.XValues = rangeOfDates

        Set butlSeries.Values = dataRange.Columns(chartIndex)

        Set butlSeries.name = titleRange


        Set targetSeries = butlSeries.AddSeriesToChart(targetChart)


        targetSeries.MarkerSize = MARKER_SIZE

        targetSeries.MarkerStyle = xlMarkerStyleCircle


        chartIndex = chartIndex + 1


    Next titleRange


    Application.ScreenUpdating = True

End Sub
```

applying common formatting to all Charts

ChartDefaultFormat.md

```
Public Sub ChartDefaultFormat()
```

```

    Const MARKER_SIZE As Long = 3
    Dim majorGridlineColor As Long
    majorGridlineColor = RGB(242, 242, 242)
    Const TITLE_FONT_SIZE As Long = 12
    Const SERIES_LINE_WEIGHT As Single = 1.5

    Dim targetObject As ChartObject

    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        Dim targetChart As Chart

        Set targetChart = targetObject.Chart

        Dim targetSeries As series
        For Each targetSeries In targetChart.SeriesCollection

            targetSeries.MarkerSize = MARKER_SIZE
            targetSeries.MarkerStyle = xlMarkerStyleCircle

            If targetSeries.ChartType = xlXYScatterLines Then targetSeries.
                Format.Line.Weight = SERIES_LINE_WEIGHT

            targetSeries.MarkerForegroundColorIndex = xlColorIndexNone
            targetSeries.MarkerBackgroundColorIndex = xlColorIndexAutomatic

        Next targetSeries


        targetChart.HasLegend = True
        targetChart.Legend.Position = xlLegendPositionBottom

        Dim targetAxis As Axis
        Set targetAxis = targetChart.Axes(xlValue)

        targetAxis.MajorGridlines.Border.Color = majorGridlineColor
        targetAxis.Crosses = xlAxisCrossesMinimum

        Set targetAxis = targetChart.Axes(xlCategory)

        targetAxis.HasMajorGridlines = True

        targetAxis.MajorGridlines.Border.Color = majorGridlineColor

        If targetChart.HasTitle Then
            targetChart.ChartTitle.Characters.Font.Size = TITLE_FONT_SIZE
            targetChart.ChartTitle.Characters.Font.Bold = True
        End If

        Set targetAxis = targetChart.Axes(xlCategory)

    Next targetObject

End Sub
```

Chart_AddTitles.md

```
Public Sub Chart_AddTitles()

    Dim targetObject As ChartObject
    Const X_AXIS_TITLE As String = "x axis"
    Const Y_AXIS_TITLE As String = "y axis"
    Const SECOND_Y_AXIS_TITLE As String = "2and y axis"
    Const CHART_TITLE As String = "chart"

    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        With targetObject.Chart
            If Not .Axes(xlCategory).HasTitle Then
                .Axes(xlCategory).HasTitle = True
                .Axes(xlCategory).AxisTitle.Text = X_AXIS_TITLE
            End If

            If Not .Axes(xlValue, xlPrimary).HasTitle Then
                .Axes(xlValue).HasTitle = True
                .Axes(xlValue).AxisTitle.Text = Y_AXIS_TITLE
            End If

            '2015 12 14, add support for 2and y axis
            If .Axes.Count = 3 Then
                If Not .Axes(xlValue, xlSecondary).HasTitle Then
                    .Axes(xlValue, xlSecondary).HasTitle = True
                    .Axes(xlValue, xlSecondary).AxisTitle.Text =
                       SECOND_Y_AXIS_TITLE
                End If
            End If

            If Not .HasTitle Then
                .HasTitle = True
                .ChartTitle.Text = CHART_TITLE
            End If
        End With
    Next targetObject

End Sub
```

Chart_ApplyFormattingToSelected.md

```
Public Sub Chart_ApplyFormattingToSelected()

    Dim targetObject As ChartObject
    Const MARKER_SIZE As Long = 5

    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        Dim targetSeries As series

        For Each targetSeries In targetObject.Chart.SeriesCollection
            targetSeries.MarkerSize = MARKER_SIZE
        Next targetSeries
    Next targetObject

End Sub
```

Chart_ApplyTrendColors.md

```
Public Sub Chart_ApplyTrendColors()

    Dim targetObject As ChartObject
    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        Dim targetSeries As series
        For Each targetSeries In targetObject.Chart.SeriesCollection

            Dim butlSeries As New bUTLChartSeries
            butlSeries.UpdateFromChartSeries targetSeries

            targetSeries.MarkerForegroundColorIndex = xlColorIndexNone
            targetSeries.MarkerBackgroundColor = Chart_GetColor(butlSeries.
                SeriesNumber)

            targetSeries.Format.Line.ForeColor.RGB = targetSeries.
                MarkerBackgroundColor

        Next targetSeries
    Next targetObject
End Sub
```

Chart_CreateChartWithSeriesForEachColumn.md

```
Public Sub Chart_CreateChartWithSeriesForEachColumn()
    'will create a chart that includes a series with no x value for each
       column

    Dim dataRange As Range
    Set dataRange = GetInputOrSelection("Select chart data")

    'create a chart
    Dim targetObject As ChartObject
    Set targetObject = ActiveSheet.ChartObjects.Add(0, 0, 300, 300)

    targetObject.Chart.ChartType = xlXYScatter

    Dim targetColumn As Range
    For Each targetColumn In dataRange.Columns

        Dim chartDataRange As Range
        Set chartDataRange = RangeEnd(targetColumn.Cells(1, 1), xlDown)

        Dim butlSeries As New bUTLChartSeries
        Set butlSeries.Values = chartDataRange

        butlSeries.AddSeriesToChart targetObject.Chart
    Next targetColumn

End Sub
```

Chart_CreateDataLabels.md

```
Public Sub Chart_CreateDataLabels()

    Dim targetObject As ChartObject
    On Error GoTo Chart_CreateDataLabels_Error

    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        Dim targetSeries As series
        For Each targetSeries In targetObject.Chart.SeriesCollection

            Dim dataPoint As Point
            Set dataPoint = targetSeries.Points(2)

            dataPoint.HasDataLabel = False
            dataPoint.DataLabel.Position = xlLabelPositionRight
            dataPoint.DataLabel.ShowSeriesName = True
            dataPoint.DataLabel.ShowValue = False
            dataPoint.DataLabel.ShowCategoryName = False
            dataPoint.DataLabel.ShowLegendKey = True

        Next targetSeries
    Next targetObject

    On Error GoTo 0
    Exit Sub

Chart_CreateDataLabels_Error:

    MsgBox "Error " & Err.Number & " (" & Err.Description & ") in procedure
        Chart_CreateDataLabels of Module Chart_Format"

End Sub
```

Chart_ExtendSeriesToRanges.md

```
Public Sub Chart_ExtendSeriesToRanges()

    Dim targetObject As ChartObject

    For Each targetObject In Chart_GetObjectsFromObject(Selection)

```

```
        Dim targetSeries As series

        'get each series
        For Each targetSeries In targetObject.Chart.SeriesCollection

            'create the bUTL obj and manipulate series ranges
            Dim butlSeries As New bUTLChartSeries
            butlSeries.UpdateFromChartSeries targetSeries

            If Not butlSeries.XValues Is Nothing Then
                targetSeries.XValues = RangeEnd(butlSeries.XValues.Cells(1),
                    xlDown)
            End If
            targetSeries.Values = RangeEnd(butlSeries.Values.Cells(1), xlDown
                )

        Next targetSeries
    Next targetObject
End Sub
```

Chart_GoToXRange.md

```
Public Sub Chart_GoToXRange()


    If TypeName(Selection) = "Series" Then
        Dim b As New bUTLChartSeries
        b.UpdateFromChartSeries Selection

        b.XValues.Parent.Activate
        b.XValues.Activate
    Else
        MsgBox "Select a series in order to use this."
    End If

End Sub
```

Chart_SortSeriesByName.md

```
Public Sub Chart_SortSeriesByName()
    'this will sort series by names
    Dim targetObject As ChartObject
    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        'uses a simple bubble sort but it works... shouldn't have 1000 series
            anyways
        Dim firstChartIndex As Long
        Dim secondChartIndex As Long
        For firstChartIndex = 1 To targetObject.Chart.SeriesCollection.Count
            For secondChartIndex = (firstChartIndex + 1) To targetObject.
                Chart.SeriesCollection.Count

                Dim butlSeries1 As New bUTLChartSeries
                Dim butlSeries2 As New bUTLChartSeries

                butlSeries1.UpdateFromChartSeries targetObject.Chart.
                    SeriesCollection(firstChartIndex)
                butlSeries2.UpdateFromChartSeries targetObject.Chart.SeriesCollection(secondChartIndex)

                If butlSeries1.name.Value > butlSeries2.name.Value Then
                    Dim indexSeriesSwap As Long
                    indexSeriesSwap = butlSeries2.SeriesNumber
                    butlSeries2.SeriesNumber = butlSeries1.SeriesNumber
                    butlSeries1.SeriesNumber = indexSeriesSwap
                    butlSeries2.UpdateSeriesWithNewValues
                    butlSeries1.UpdateSeriesWithNewValues
                End If

            Next secondChartIndex
        Next firstChartIndex
    Next targetObject
End Sub
```

ChartFlipXYValues.md

```
Public Sub ChartFlipXYValues()

    Dim targetObject As ChartObject
    Dim targetChart As Chart
    For Each targetObject In Chart_GetObjectsFromObject(Selection)
        Set targetChart = targetObject.Chart

        Dim butlSeriesies As New Collection
        Dim butlSeries As bUTLChartSeries

        Dim targetSeries As series
        For Each targetSeries In targetChart.SeriesCollection
            Set butlSeries = New bUTLChartSeries
            butlSeries.UpdateFromChartSeries targetSeries

            Dim dummyRange As Range

            Set dummyRange = butlSeries.Values
            Set butlSeries.Values = butlSeries.XValues
            Set butlSeries.XValues = dummyRange

            'need to change the series name also
            'assume that title is same offset
            'code blocked for now
            If False And Not butlSeries.name Is Nothing Then
                Dim rowsOffset As Long, columnsOffset As Long
                rowsOffset = butlSeries.name.Row - butlSeries.XValues.Cells(1, 1).Row
                columnsOffset = butlSeries.name.Column - butlSeries.XValues.Cells(1, 1).Column

                Set butlSeries.name = butlSeries.Values.Cells(1, 1).Offset(rowsOffset, columnsOffset)
            End If

            butlSeries.UpdateSeriesWithNewValues

        Next targetSeries
```

```vba

        ''need to flip axis labels if they exist
        ''three cases: X only, Y only, X and Y

        If targetChart.Axes(xlCategory).HasTitle And Not targetChart.Axes(
            xlValue).HasTitle Then

            targetChart.Axes(xlValue).HasTitle = True
            targetChart.Axes(xlValue).AxisTitle.Text = targetChart.Axes(
                xlCategory).AxisTitle.Text
            targetChart.Axes(xlCategory).HasTitle = False

        ElseIf Not targetChart.Axes(xlCategory).HasTitle And targetChart.Axes
            (xlValue).HasTitle Then
            targetChart.Axes(xlCategory).HasTitle = True
            targetChart.Axes(xlCategory).AxisTitle.Text = targetChart.Axes(
                xlValue).AxisTitle.Text
            targetChart.Axes(xlValue).HasTitle = False

        ElseIf targetChart.Axes(xlCategory).HasTitle And targetChart.Axes(
            xlValue).HasTitle Then
            Dim swapText As String

            swapText = targetChart.Axes(xlCategory).AxisTitle.Text

            targetChart.Axes(xlCategory).AxisTitle.Text = targetChart.Axes(
                xlValue).AxisTitle.Text
            targetChart.Axes(xlValue).AxisTitle.Text = swapText

        End If

        Set butlSeriesies = Nothing

    Next targetObject

End Sub
```

ChartMergeSeries.md

```
Public Sub ChartMergeSeries()

    Dim targetObject As ChartObject
    Dim targetChart As Chart
    Dim firstChart As Chart

    Dim isFirstChart As Boolean
    isFirstChart = True

    Application.ScreenUpdating = False

    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        Set targetChart = targetObject.Chart
        If isFirstChart Then
            Set firstChart = targetChart
            isFirstChart = False
        Else
```

```
            Dim targetSeries As series
            For Each targetSeries In targetChart.SeriesCollection

                Dim newChartSeries As series
                Dim butlSeries As New bUTLChartSeries

                butlSeries.UpdateFromChartSeries targetSeries
                Set newChartSeries = butlSeries.AddSeriesToChart(firstChart)

                newChartSeries.MarkerSize = targetSeries.MarkerSize
                newChartSeries.MarkerStyle = targetSeries.MarkerStyle

                targetSeries.Delete

            Next targetSeries

            targetObject.Delete

        End If
```

```
    Next targetObject

    Application.ScreenUpdating = True

End Sub
```

ChartSplitSeries.md

```
Public Sub ChartSplitSeries()

    Dim targetObject As ChartObject
    Dim targetChart As Chart

    Dim targetSeries As series
    For Each targetObject In Chart_GetObjectsFromObject(Selection)

        For Each targetSeries In targetObject.Chart.SeriesCollection

            Dim newChartObject As ChartObject
            Set newChartObject = ActiveSheet.ChartObjects.Add(0, 0, 300, 300)

            Dim newChartSeries As series
            Dim butlSeries As New bUTLChartSeries

            butlSeries.UpdateFromChartSeries targetSeries
            Set newChartSeries = butlSeries.AddSeriesToChart(newChartObject.Chart)

            newChartSeries.MarkerSize = targetSeries.MarkerSize
            newChartSeries.MarkerStyle = targetSeries.MarkerStyle

            targetSeries.Delete

        Next targetSeries


        targetObject.Delete

```

```
    Next targetObject
End Sub
```

DeleteAllCharts.md

```
Public Sub DeleteAllCharts()

    If MsgBox("Delete all charts?", vbYesNo) = vbYes Then
        Application.ScreenUpdating = False

        Dim chartObjectIndex As Long
        For chartObjectIndex = ActiveSheet.ChartObjects.Count To 1 Step -1

            ActiveSheet.ChartObjects(chartObjectIndex).Delete

        Next chartObjectIndex

        Application.ScreenUpdating = True

    End If
End Sub
```

RemoveZeroValueDataLabel.md

```
Public Sub RemoveZeroValueDataLabel()

    'uses the ActiveChart, be sure a chart is selected
    Dim targetChart As Chart
    Set targetChart = ActiveChart

    Dim targetSeries As series
    For Each targetSeries In targetChart.SeriesCollection

        Dim seriesValues As Variant
        seriesValues = targetSeries.Values

        'include this line if you want to reestablish labels before deleting
        targetSeries.ApplyDataLabels xlDataLabelsShowLabel, , , , True, False
            , False, False, False

        'loop through values and delete 0-value labels
        Dim pointIndex As Long
        For pointIndex = LBound(seriesValues) To UBound(seriesValues)
            If seriesValues(pointIndex) = 0 Then
                With targetSeries.Points(pointIndex)
                    If .HasDataLabel Then .DataLabel.Delete
                End With
            End If
        Next pointIndex
    Next targetSeries
End Sub
```

UpdateFromChartSeries.md

```
Public Sub UpdateFromChartSeries(targetSeries As series)

```

```
'this will work for the simple case where all items are references
Const FIND_STRING As String = "SERIES("
Const COMMA As String = ","
Const CLOSE_BRACKET As String = ")"

Set series = targetSeries

Dim targetForm As Variant

'=SERIES("Y",Sheet1!$C$8:$C$13,Sheet1!$D$8:$D$13,1)

'pull in the formula
targetForm = targetSeries.Formula

'uppercase to remove match errors
targetForm = UCase(targetForm)

'remove the front of the formula
```

```
    targetForm = Replace(targetForm, FIND_STRING, vbNullString)

    'find the first foundPosition
    Dim foundPosition As Long
    foundPosition = InStr(targetForm, COMMA)

    If foundPosition > 1 Then
        'need to catch an error here if a text name is used instead of a
            valid range
        On Error Resume Next
        Set Me.name = Range(left(targetForm, foundPosition - 1))
        If Err <> 0 Then pName = left(targetForm, foundPosition - 1)
        On Error GoTo 0
    End If

    'pull out the title from that
    targetForm = Mid(targetForm, foundPosition + 1)

    foundPosition = InStr(targetForm, COMMA)
```

```vba

    If foundPosition > 1 Then Set Me.XValues = Range(left(targetForm,
        foundPosition - 1))

    targetForm = Mid(targetForm, foundPosition + 1)

    foundPosition = InStr(targetForm, COMMA)
    Set Me.Values = Range(left(targetForm, foundPosition - 1))
    targetForm = Mid(targetForm, foundPosition + 1)

    foundPosition = InStr(targetForm, CLOSE_BRACKET)
    Me.SeriesNumber = left(targetForm, foundPosition - 1)

    Me.ChartType = targetSeries.ChartType
End Sub
```