# 3.bölüm

**3**

# dört temel nesne

Excel, Nesne Modeline dayanan bir yapıya sahiptir demiştik. Bu, şu demek: Excel’deki herşey bir nesnedir ve bu nesneler hiyerarşik bir yapı içinde bulunurlar. Çok sayıda nesne olmakla birlikte en sık kullanacağımız nesneler aşağıdaki dörtlüdür.

* Range
* Worksheet
* Workbook
* Application

NOT: Eğer bölümlerde sırayla ilerleyen biriyseniz, artık gözünüz aydın diyebilirim, çünkü bu bölümden itibaren kod yazımında oldukça ivme kazanacağız. O yüzden nerdeyse tüm makrolarda bulunan koşullu yapıları ve döngüleri de sık sık kullanacağız. VBA yerine VB.NET anlatıyor olsaydım bunları daha önce anlatırdım. Ancak VBA olunca işin rengi biraz değişiyor. O yüzden önce bu konulara bakın demek istemiyorum, çünkü oradaki şeyleri anlamak için de bu dört nesneyi de az çok tanımak gerekiyor. O yüzden bir oraya bir buraya bakarak da ilerleme yolunu tercih edebilirsiniz.

## Range Nesnesi

Range nesnesi, VBA dünyasındaki en temel nesnemizdir. Kodlarımızın nerdeyse hepsinde bu nesneyi kullanacağız ve dahası, kodlarımızın önemli bir kısmını bu nesne ve türevleri oluşturacak. Bu nedenle bu nesnenin üyelerini yani metod ve özelliklerini(property) çok iyi bilmemiz gerekiyor.

### Temeller

#### Hücrelere erişmek

Range diyince aklımıza neler gelir: Tek bir hücre, bitişik olup olmaması fark etmeyen hücre grubu, bir satır, birkaç satır, veya sütunlar. Hemen örneklere bakalım:

|  |  |  |
| --- | --- | --- |
| Gösterim | Tür | Anlamı |
| Range("A1") | Tek hücre | A1 hücresi |
| Range("A1: B2") | Bitişik çoklu hücre | A1 ile B2 arası |
| Range("A1: B2,C3: D4") | Bitişik olmayan çoklu hücre | A1-B2 ve C3-D4 arası |
| Range("Ozelalan") | İsim verilmiş alan | Name Managerde "Ozelalan" olarak belirlenen yer |
| Range(Range("A1"),range("C5")) | başlangıç ve bitişi ayrı belirtilmiş rangeler | A1-C5 arası |

Bir hücreyi işaret etmenin başka yolları da var. Özellikle döngülü kodlarda Cells(satırno, sütunno) veya Cells(satırno, sütunharfi) ifadelerini çok kullanacağız. Bunun dışında bir de []'ler içinde direkt hücre adresini vermek şeklinde de ulaşacağız. Ör:

Cells(3,2).Select 'B3 hücresini seçer

Cells(3,"B").Select 'Bu da B3 hücresini seçer

[A1: B5].Select 'A1-B5 arasını seçer

[ozelalan].Select

Bu arada gördüğünüz üzere bir Range'i seçmek için Select metodunu kullanıyoruz.

'Range ile döngü

For i = 1 To 10

Range("A" & i).Value = i & ".satır"

Next

'Cells ile döngü

For i = 1 to 10

Cells(i,1).Value= i & ".satır"

Next

Şimdi de tüm sütun seçme işlemleri var

|  |  |  |
| --- | --- | --- |
| Gösterim | Tür | Anlamı |
| Range("A: A") | Range içinde harf | A kolonu |
| Columns("A") | Columns içinde harf | A kolonu |
| Columns(1) | Columns içinde index | A kolonu |
| Range("A: C") | Bitişik çoklu kolon | A-C arası kolonlar |
| Columns("A: C") | Bitişik çoklu kolon | A-C arası kolonlar |
| Range("A: C,E: E,H: K") | Bitişik olmayan çoklu kolon sadece Range ile | A-C, E ve H-K kolonları |
| Columns | Tüm kolonlar | Tüm kolonlar |

Bir de tüm satır seçme işlemleri var

|  |  |  |
| --- | --- | --- |
| Gösterim | Tür | Anlamı |
| Range("1: 1") | Range içinde satır no | 1.satır |
| Rows(1) | Rows içinde index | 1.satır |
| Range("1: 5") | Bitişik çoklu satır | 1-5 arası satırlar |
| Rows("1: 5") | Bitişik çoklu satır | 1-5 arası satırlar |
| Range("1: 5,8: 10") | Bitişik olmayan çoklu satır sadece Range ile | 1-5 ve 8-10 arası satırlar |
| Rows | Tüm satırlar | Tüm satırlar |

Çoklu seçimlerde bir de Union metodu kullanılır. Gerçi bu metod Range nesnesinin değil Application nesnesinin metodudur ama olsun yeri geldiği için burada değinmiş olduk. (Kesişim kümesini bulan Intersect metodunu ise Applicaton konusunda inceleyeceğiz.)

Dim r1, r2, myMultipleRange As Range

Set r1 = Sheets("Sheet1").Range("A1: B2")

Set r2 = Sheets("Sheet1").Range("C3: D4")

Set myMultipleRange = Union(r1, r2)

myMultipleRange.Font.Bold = True

Bir yerde çoklu seçim olup olmadığını anlamak için Areas özelliği kullanılır.

If Selection.Areas.Count > 1 Then

MsgBox "Çoklu seçim var, lütfen tek bir alan seçip öyle devam edin"

Exit Sub

End If

Bazı durumlarda Range'leri bir değişkene atayacağız. Range nesnesini, adı üstünde bir nesne olduğu için Set ifadesi ile atamasını yaparız.

Dim alan As Range

Set alan=Range("A1: B5")

alan.Formula="=Rand()"

#### Hücreleri seçmek

Rangeler aslında bir Worksheet’in Range propertysinin dönen değeridir. O yüzden bir sayfa ismiyle kullanılırlar, ancak çoğu zaman sayfa ismi olmadan kullanırız. Bunun anlamı, o anki aktif sayfanın Rangeler’i üzerinde işlem yapıyoruz demektir. Yani aslında **Range("A1")** yazmak, **ActiveSheet.Range("A1")** yazmanın kolay yoludur.

Yukarıdaki örneklerde gördüğünüz üzere range nesnelerini seçmek için Select metodunu kullanıyoruz. Range("A1").Select gibi.

Bunu kullanırken dikkat edilecek nokta, seçtiğiniz Range nesnesinin aktif sayfada olmasıdır. Yani 2.sayfada iken şöyle bir seçim yapamazsınız.

Sheets(1).Range("A1").Select

Bunu tek satırda yapmanın farklı bir yolu var: (Biraz garip ama doğrusu bu)

Application.Goto Sheets(1).Range("A1")

'veya bir diğer yol da önce ilgili sayfayı seçip sonra hücreyi seçmek olabilir

Sheets(1).Select

Range("A1").Select

**NOT**: Excel, aynı anda birden çok sayfadaki Range'i seçmemize izin vermez.

##### Seçme işleminin performansa etkisi

Select işlemi oldukça maliyetli bir işlemdir. Bu nedenle mecbur kalınmadığı sürece seçme işlemi yapılmamalıdır. Örneğin bir hücreye değer atanacaksa seçmeden de atanabilir. Aşağıdaki örnek ne demek istediğimi net şekilde anlatmaktadır.

Sub secim\_maliyeti()

Dim bas As Single, bitis As Single

bas = Timer

For i = 1 To 10000

Cells(i, 1).Select 'bu varken 43 sn, yokken 1 sn

Cells(i, 1) = i

Next i

bitis = Timer

Debug.Print bitis - bas

End Sub

##### Select vs Activate

Select'e benzer bir görevi olan bir de Activate metodu vardır, ki **Activate sadece tek bir hücreyi aktive ederken Select ile bir hücre grubunu da seçebiliriz.** Üstelik hali hazırda seçili bir yer varken, seçim değiştirilmeden o seçim içinde bir hücre bile aktive edilebilir. Aşağıda bu konuyla ilgili küçük bir örneğimiz olacak, ancak yine bu konuyla alakalı olarak ActiveCell ve Selection farkına da değinelim.

##### ActiveCell vs Selection

O anda bulunduğumuz hücre üzerinde bir işlem yapmak istersek bir Range türü olan **ActiveCell** nesnesini kullanırız.

Bulunduğunuz yer tek bir hücre değil de hücre grubu ise **Selection** nesnesini kullanırız.

Bir hücre grubu seçiliyken ActiveCell özelliği kullanılırsa o hücre grubunun ilk hücresi(sol üstteki) dikkate alınır.

Bunların ikisi de **Application** nesnesinin bir propertysi olup, Application ifadesi olmadan da kullanılabilirler(Application default nesne olduğu için). Bu arada bu iki özellik de Range objesi döndürdükleri için bunlardan nesne diye bahsederiz.

Sub select\_activate()

Range("A1: B10").Select

Debug.Print ActiveCell.Address 'sol üstteki ilk hücre, yani A1

Debug.Print Selection.Address 'tüm alan, yani A1: b10

Range("B8").Activate

Debug.Print ActiveCell.Address 'B8

Debug.Print Selection.Address 'seçili hücre B8 olmakla birlikte seçili alan hala aynıdır, değişmemiştir, o yüzden Immediate Window'da a1: b10 yazar

Range("C8").Activate 'ilk seçim alanının dışında bir hücre seçiliyor, artık selection da değişmiştir

Debug.Print ActiveCell.Address 'C8

Debug.Print Selection.Address 'C8

End Sub

##### Areas

Ctrl tuşuyla seçilen ve birbirinden farklı bölgelerde bulunan hücrelere Areas collection'ı ile ulaşırız.

Sub areasornek()

x = 10

For Each alan In Selection.Areas

alan.Interior.ColorIndex = x

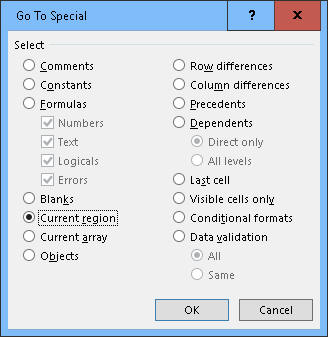
x = x + 1

Next alan

End Sub

##### CurrentRegion ve UsedRange

Bulunduğunuz hücrenin veya o anki aktif seçili bölgenin etrafındaki tüm bitişik hücrelerden oluşan bölgeyi seçmek, o bölgede işlem yapmak için CurrentRegion özelliği kullanılır. Aynı işlemi Excel’de **Home>Editing>Find&Select>Go To Speacial** seçeneğinden aşağıdaki gibi de yapabilirsiniz.



Syntax: RangeNesnesi.CurrentRegion şeklinde olup örnek kullanımı şöyle olabilir.

ActiveCell.CurrentRegion.Select

Peki bu örnek yeterli değil. O zaman size bir ipucu. Sıralama ve filtreleme işlemlerinizi Makro Kaydedici ile yaptığınızda , VBA'in size ürettiği kodda sabit bir alan görebilirsiniz. Ancak sizin makronuz her zaman bu sabit alan üzerinde çalışmayacaktır. O yüzden o sabit alanı CurrentRegion özelliğinden faydalanarak değiştirebilirsiniz. Ama bunu yapmak o kadar da basit değil. Offset ve Resize özelliklerinden de faydalanmamız gerekecek. Bu örneği biraz aşağıda ilgili yere gelince düzelteceğiz.

Range("A2").Select

ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Clear

ActiveWorkbook.Worksheets("Sheet1").Sort.SortFields.Add Key: =Range( \_

"A2"), SortOn: =xlSortOnValues, Order: =xlAscending, DataOption: = \_

xlSortNormal

With ActiveWorkbook.Worksheets("Sheet1").Sort

.SetRange Range("A2: F175") 'işte burayı değiştireceğiz, Range("A2: F175") yerine Range("A1").CurrentRegion yazıp deneyin, işe yaramadığını görün

.Header = xlNo

.MatchCase = False

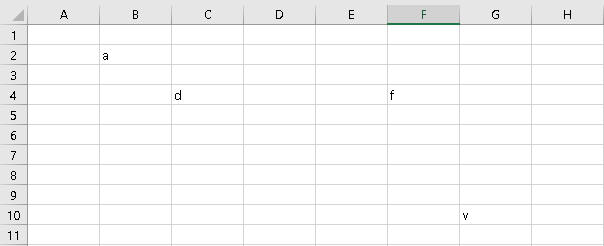
.Orientation = xlTopToBottom

.SortMethod = xlPinYin

.Apply

End With

UsedRange, Range nesnesinin değil Worksheet nesnesinin bir özeliğidir ama CurrentRegion’a benzerliği nedeniyle burada bahsetmenin uygun olacağını düşündüm. Bununla birbirine komşu olmayan alanları bile kapsayacak şekilde, tüm işlenmiş alanları seçmek için kullanılır. Burada işlenmişten kastım, bir veri girişi yapılmış veya default formatı değiştirilmiş bir hücredir. Mesela aşağıdaki örnekte, UsedRange seçimi yapıldığında B2: G10 seçilirken



H11 hücresinin font büyüklüğü 1 birim artırılırsa veya Tarih formatı uygulanırsa, içi boş bile olsa UsedRange bunu da kapsayacak şekilde genişler ve B2: H11 olur.

Syntax: WorkSheet.UsedRange şeklinde olup örnek kullanımı da şöyledir.

ActiveSheet.UsedRange.Interior.Color = vbYellow

##### Özel seçimler(SpecialCells)

Bazen de boş hücreler, formül içeren hücreler, sadece görünen hücreler gibi özel seçimler yapmak isteriz. Bunlar için de SpecialCells özelliği kullanılır. İki parametre alır ve syntaxı şöyledir.

Syntax: SpecialCells(Type, Value)

Type parametresinin alacağı değerler şöyledir:

|  |  |  |
| --- | --- | --- |
| Constant | Anlamı | Numerik Değeri |
| xlCellTypeAllFormatConditions. | Herhangibir formatı olan | -4172 |
| xlCellTypeAllValidation. | Validation kuralı eklenmiş hücreler | -4174 |
| xlCellTypeBlanks. | Boş hücreler | 4 |
| xlCellTypeComments. | Yorumlu hücreler | -4144 |
| xlCellTypeConstants. | Sabit değer içeren hücreler(formülsüzler yani) | 2 |
| xlCellTypeFormulas. | Formüllü hücreler | -4123 |
| xlCellTypeLastCell. | Ctrl+End etkisi(Son hücre) | 11 |
| xlCellTypeSameFormatConditions. | Aynı conditional formatı olan hücreler | -4173 |
| xlCellTypeSameValidation. | aynı validation kuralı olan hücreler | -4175 |
| xlCellTypeVisible. | Görünür hücreler | 12 |

Eğer tip olarak constant veya formül seçildiyse ikinci parametre olarak şunlar seçilebilir.

|  |  |
| --- | --- |
| Constant | Numerik Değeri |
| xlErrors | 16 |
| xlLogical | 4 |
| xlNumbers | 1 |
| xlTextValues | 2 |

Birkaç örnek vermek gerekirse;

'nümerik değer içeren hücrelerin arkaplan rengini sarı yapalım

Worksheets("Sheet1").Cells.SpecialCells(xlCellTypeConstants, xlNumbers).Interior.Color = vbYellow

'formül içeren hücrelerin yazı rengini kırmızı yapalım

Worksheets("Sheet1").Cells.SpecialCells(xlCellTypeConstants, xlTextValues).Font.Color = vbRed

##### Uç noktalar

Bir hücrenin veya hücre grubunun sanki Excel'de Ctrl+Home, Ctrl+End, Ctrl+Ok tuşlarına basılmış gibi bir etki göstermesi için de çeşitli propertyler var. Şimdi onlara bakalım.

|  |  |
| --- | --- |
| Eylem | Yöntem |
| Ctrl+Home | Cells.SpecialCells(xlCellTypeVisible)(1).Select \* |
| Ctrl+End | Cells.SpecialCells(xlCellTypeLastCell).Select |
| Ctrl+↑ | ActiveCell.End(xlUp).Select |
| CTrl+↓ | ActiveCell.End(xlDown).Select |
| CTrl+→ | ActiveCell.End(xlToRight).Select |
| CTrl+← | ActiveCell.End(xlToLeft).Select |

\*Gizlenmiş/Filtrelenmiş bir satır/sütun yoksa Range("A1").Select de bu işi görecektir

#### Bir hücrenin değerini okumak, ona değer atamak

Range nesnesinin Value, Value2 ve Text propertyleri vardır. Bunlardan Value, bu nesnenin default özelliğidir. Default özellik şu demek, onu yazmadan da kullanabilirsiniz. Mesela **Range("A1")** ile **Range("A1").Value** tamamen özdeştir. Ancak ben şahsen sizlere iyi bir kodlamacı özelliği olarak default özellikleri es geçmeden açıkça yazmanızı öneririm.

**Value** özelliği hem okunabilir hem değer atanabilir bir özelliktir, dönüş değeri Varianttır. Yani her şeyi döndürebilir, string, integer, tarih, boş.

Dim v As Variant

v=Range("A1").Value 'Değeri okudum, Value özelliğini belirttim

Range("A1")="Volkan" 'Değer atadım, Value özelliğini belirtmedim, çünkü default özellik

**Value2** özelliği Value'ya benzer, ancak Value hücrenin içindeki gerçek değeri aynen verirken Value2 ise Tarih veya parabirimi formatındaki veriyi de düz sayıya çevirir.

**Text** ise Excel’de gözümüzle ne görüyorsak bize onu verir, yani formatlanmış halini verir.

Eğer düz bir metin veya sayı ile çalışıyorsanız Value2'yi kullanmanızı tavsiye ederim. İçlerinde en hızlısı ve sorunsuzu budur.

**Ör**: A1 hücresinde 21.01.1979 değeri varken hücre formatını Long Date olarak değiştirdikten sonra aşağıdaki kodları çalıştıralım ve farkı görelim.

Sub valuetext()

With Range("A1")

Debug.Print .Text '21 Ocak 1979 Pazar

Debug.Print .Value '21 Ocak 1979

Debug.Print .Value2 '28876

End With

End Sub

#### Hücrenin rengi

Hücrenin arka planı ve yazı rengini belirlemek için sırasıyla aşağıdaki kodları kullanırız.

Range("a1").Interior.Color = vbRed

Range("a1").Font.Color = vbYellow

'Interior ve Font propertylerinde başka neler var, şöyle bir gözatın

'işinize yarayacak neler var aklınızda tutun, ilerde lazım olabilir

#### Cut, Copy, Paste, Insert işlemleri

##### Cut, Copy,Paste

Excel’de yaptığımız Cut, Copy işlemleri için Range nesnesinin aynı isimli metotlarını kullanıyoruz. Paste işlemi içinse Range nesnesinin doğrudan bir Paste metodu yok, PasteSpecial metodu vardır. Excel’deki "Sağ tık>Yapıştır" veya Ctrl+V kombinasyonlarıyla yaptığımız yapıştırma işleminin VBA karşılığı ActiveSheet metodunun Paste metodu olup detayını Worksheet konusunda okuyabilirsiniz, aşağıda da küçük bir örnek bulunuyor.

Sub cutcopypaste()

Sheets(1).Select

Range("A5: B5").Select

Selection.Cut 'veya Selection.Copy

Sheets(2).Select

ActiveSheet.Paste

End Sub

##### PasteSpecial

A5: B5 hücre grubunda formüller varsa ve bunların sadece değerlerini yapıştırmak istersek işte o zaman PasteSpecial kullanırız. Aşağıda Macro Recorder ile kaydedilmiş bir kodu görüyoruz.

Sub cutcopypaste()

Sheets(1).Select

Range("A5: B5").Select

Selection.Copy

Sheets(2).Select

ActiveCell.PasteSpecial xlPasteValues

End Sub

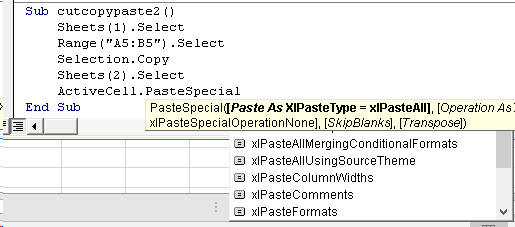
Aslında bu işlemi yapmanın basit bir yolu daha var, Copy metodunu destination parametresi ile kullanmak:

Sub cutcopypaste()

Range("A5: B5").Copy Destination: =Sheets(2).Range("A15") 'Destination tek argüman olduğ için yazmaya gerek yok

End Sub

PasteSpecial'in parametrelerinin alabileceği değerleri görmek için PasteSpecial yazdıktan sonra boşluk veya "(" tuşuna basar basmaz Intellisense bize enumeration listesini çıkarır.



**NOT:** Tek hücre için PasteSpecial'ı xlPasteValues parametresiyle yapmak yerine kısayol olarak doğrudan hücrelerin içeriğini eşitlenebilir.

Range("A2").Value=Range("A1").Value

##### Insert

Excel’de yaptığımız satırları/sütunları kesip/kopyalayıp başka bir satırların/sütunların arasına sokma işlemini yine aynı isimli Insert metodu ile yapıyoruz.

Rows("2: 2").Select

Selection.Cut

Rows("9: 9").Select

Selection.Insert Shift: =xlDown 'parametre belirtilmezse Excel kendi karar verir

#### Formül işlemleri

Biz genel olarak kodlarımızda hücrelere formül yazdırmayacağız, ancak ender de olsa bu işlemi yaptırmamız gerekebilecektir. Bunun için önce Macro Recorder ile hücrelere bir şeyler yazalım ve nasıl formül ürettiğine bakalım. Gördüğünüz gibi tüm formüller, hatta metin ifadeler bile **FormulaR1C1** özelliği ile ele alındı.

Range("H6").Select

ActiveCell.FormulaR1C1 = "volkan"

ActiveCell.FormulaR1C1 = "=TODAY()"

ActiveCell.FormulaR1C1 = "=RC[-2]\*2" 'solundaki 2 hücreye referans

Biz Excel'de formüllerimizi klasik stilde(A1 stili) yazmaya alışık olduğumuz için bu FormlaR1C1 yerine **Formula** propertysini kullanırız.

Selection.Formula = "F6\*2"

Son olarak dizi formülü yazmaya yarayan **FormulaArray** var, onun kullanımı da aşağıdaki gibi olup, Excel’deki görünümü {=MAX(IF(C: C<100;C: C))} şeklindedir.

Selection.FormulaArray = "=MAX(IF(C[-5]<100,C[-5]))"

### İleri seviye Range işlemleri

#### Range'in adresi

Address propertysi ile alınır. String değer döndürür.

Debug.Print ActiveCell.Address '$ işaretli mutlak adres

Debug.Print ActiveCell.Address(0,0) '$ işaretsiz göreceli adres

Address özelliğini worksheet olaylarında Target.Address şeklinde çok kullanacağız.

Keza, aktif hücrenin belirli bir adreste olup olmadığını kontrol etmek için de kullanılabilir.

If ActiveCell.Adress="B$1$" Then .....

#### Range Property'si

Şimdiye kadar Range nesnesini, Worksheet nesnesinin bir özelliği olarak kullanmış olduk. Ancak bunu bir Range nesnesinin özelliği olarak da kullanabiliriz.

İster tek bir hücre seçiliyken ister bir hücre grubu seçiliyken olsun, Range özelliğini kullanırsak, tek hücre için kendisini, **hücre grubu için sol üstteki ilk hücreyi** referans alarak yeni konum belirlemiş oluruz ve bu referansı da A1 hücresine olan göreli farkla elde ederiz. Ör: C3 hücresindeyken, Selection.Range("B1").Select dediğimizde ne olur tahmin edelim: A1'e göre B1 nerededir, bir sütun sağdadır, o yüzden C3'teyken Range("B1") dersek bir sağdaki D3 seçilir. Keza C4: F6 hücre grubu seçiliyken Selection.Range("C2").Select dersek ne olur;C2, A1'e göre 2 sağda 1 aşağıdadır. C4: F6'nın ilk hücresi de C4 olup 2 sağ ve 1 alttaki hücresi nedir, E5.

Ben şahsen Range nesnesinin Range özelliğini çok kullanmam, onun yerine Item ve Offset özellikleri daha kullanışlıdır. Hemen onlara bakalım.

#### Item, Cells, Offset, Resize Propertyleri ve göreceli başvurular

**Item**, Belirtilen range'in hangi satır sütunundaki hücresi olduğunu döndürür. Ör: Range("B3: D6") alanının 1.satır, 2.sütundaki hücresini şu şekilde elde ederiz.

Range("B3: D6").Item(1,2).Select 'C3 hücresi seçilir

İkinci parametre opsiyonel olup, tek parametre verilirse, ilgili alanın soldan sağa kaçıncı hücresinin seçileceği belirtilmiş olur.

Range("B3: D6").Item(1).Select 'sol üstteki ilk hücre yani B3 seçilir

Range("B3: D6").Item(2).Select 'C3 hücresi

Range("B3: D6").Item(5).Select 'C4 hücresi

Item özelliği Range nesnesinin default özelliği olup bunu yazmadan da kullanabiliriz.

Range("B3: D6")(1).Select 'Range("B3: D6").Item(1).Select demektir

Cells(2,3).Select 'Cells.Item(2,3).Select demektir.

Item özelliğinde ilgili Range'in dışına çıkabiliriz. Mesela 0 veya negatif rakamlarla sola ve üste, Range'in toplam alanından daha büyük rakamlarla da sağa ve aşağı doğru ilerleyebiliriz.

Range("D3: F6").Item(0,-1).Select 'B2 hücresi seçilir

Range("D3: F6")(13).Select 'D7 seçilir

**Cells** ile de Item'a benzer bir kullanımımız mevcuttur. Cells, hem Range nesnesinin, hem de Worksheet nesnesinin bir propertysidir. Range ile kullanıldığında o Range'in hangi hücresini seçeceğimiz belirtirken worksheet ile kullanırken(veya aktif sayfa için sheet adı yazmadan) tüm sayfanın kaçıncı hücresi olduğunu belirtiriz.

Bunların bir arada kullanımına ait örnekleri aşağıda bulabilirsiniz.

Sub görelisecim()

Dim alan As Range

Set alan = Range("C5: E8")

alan.Select ' tamamı

alan.Range("A1").Select

alan.Item(0, 0).Select ' bir satır sol üst

alan.Item(1, 1).Select 'sol üstteki ilk hücre

alan.Item(1).Select 'sol üstteki ilk hücre

alan.Item(0).Select 'sol üstteki ilk hücrenin bir solu

alan.Cells(1, 1).Select 'sol üstteki ilk hücre

alan.Cells.Select 'tamamı

alan.Offset(1, 1).Select 'toplam alan büyüklüğü aynı kalack şekilde bir aşağı bir sağa kaydır

alan.Offset(0, 0).Select 'tamamı

alan.Offset().Select 'tamamı

alan(1, 1).Select 'item gibi davranır

alan(0, 0).Select 'item gibi davarnır

alan(1).Select 'item gibi davranır

alan(0).Select 'item gibi davranır

End Sub

Gördüğünüz gibi Item ile Cell hep aynı sonucu veriyor. O halde neden iki ayrı property var diye düşünüyor olabilirsiniz. Cevap: Item aslında Range'e özgü bir özellik değildir. Item, collection tarzı tüm  nesnelerin genel bir özelliği olup, collectionlardaki elemanların her birini ifade eder. Range de bir hücreler koleksiyonu olduğu için bu özelliği devralmıştır.

Bu arada fark ettiniz mi bilmiyorum ama bu tek indeks verme olayı, bir sütunda aşağı doru ilerlemek için güzel bir fırsat sunuyor. Mesela Range("A1")(1) A1 hücresini ifade ederken, Range("A1")(5) A5, Range("A4")(11) de A14 hücresini. Bu yöntem az sonra göreceğimiz Offsetin güzel bir alternatifi olmaktadır.

**Offset(x,y)** ile referans verilen bir range'in x satır sağına(x negatifse soluna) ve y sütun altına(y negatifse üstüne) gideceğimize karar veririz.

Syntax: RangeObject.Offset(satır,sütun) şeklindedir.

Burda Range tek bir hücre olabildiği gibi, hücre grubu veya bir satır/sütun da olabilir

Sub offsetornek()

Range("C2").Offset(1, 0).Select 'C3

Range("C2").Offset(-1, 2).Select 'E1

Range("C2").Offset(0, -2).Select 'A2

Range("C2").Offset(0, 0).Select 'C2

Range("C2").EntireRow.Offset(1).Select '3.satır. Range("C2").EntireRow.Offset(1,0).Select ile aynıdır. İkinci parametre yoksa 0 anlamındadır

Range("C2").EntireRow.Offset(-1).Select '1.satır

Range("C2").EntireColumn.Offset(, -1).Select '2.sütun. Range("C2").EntireColumn.Offset(0, -1).Select ile aynıdır. ilk paramterde 0 yerine boş da geçilebilir

Range("C2: F6").Offset(1, 1).Select 'D3: G7 seçilir

End Sub

**Resize** ile bir Range nesnesi yeniden boyutlandırılır.

Syntax: Resize(satırboyutu, sütunboyutu). İki parametre de opsiyonel olup belirtilmezlerse olduğu yerde kalır.

Sub resizeornek1()

Range("C3: G7").Select

Selection.Resize(Selection.Rows.Count - 1, Selection.Columns.Count + 2).Select

Range("C3: G7").Resize.Select ' aynen kalır

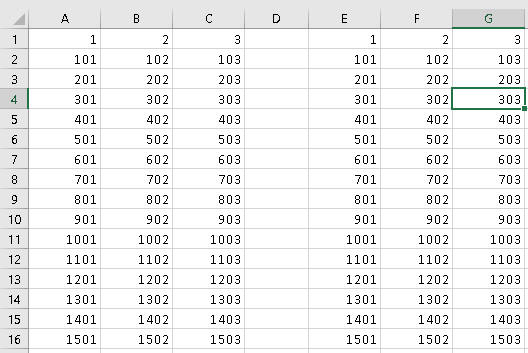
Range("C3: G7").Resize().Select 'aynen kalır

Range("C3: G7").Resize(1).Select 'kolon parametresi yok, o yüzden aynı kalır, satır ise 1 satır olacak şekilde daralır

Range("C3: G7").Resize(, 2).Select 'satır parametresi yok, o yüzden aynı kalır, sütun ise 2 sütun olacak şekilde daralır

End Sub

Resize'ın güzel kullanımlarından biri de bir range'in olduğu gibi bir diziye atanıp başka bir yere kopyalanmasındadır. Aşağıdaki örneğe bakalım. A1: C16 arasındaki herşey E1: G16 alanına aktarılacak. İşlemi, biraz aşağıda göreceğiniz “range\_diziata” koduyla yaptık.



Şimdi de, biraz yukarda CurrentRegion özelliğinde bir örneğimiz vardı, yeri gelince düzelteceğiz demiştik. Yapacağımız şey, bir listeyi başlığı hariç seçmek olacak.

enalt = Range("A1").End(xlDown).Row

Set alan = Range("A1").CurrentRegion.Resize(enalt - 1).Offset(1)

Range("A2").Select

ActiveWorkbook.Worksheets("calculatedlar").Sort.SortFields.Clear

ActiveWorkbook.Worksheets("calculatedlar").Sort.SortFields.Add Key: =Range( \_

"A2"), SortOn: =xlSortOnValues, Order: =xlAscending, DataOption: = \_

xlSortNormal

With ActiveWorkbook.Worksheets("calculatedlar").Sort

.SetRange alan 'Makro recordardan sabit gelen bu kısmı değiştirdik

.Header = xlNo

.MatchCase = False

.Orientation = xlTopToBottom

.SortMethod = xlPinYin

.Apply

End With

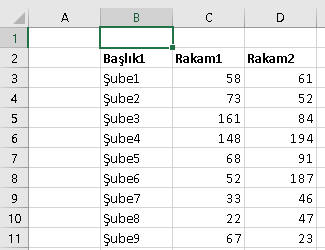
#### Column, Columns, Row, Rows

**Columns**: Bir Range nesnesinin kolonlarını ifade eder. Genelde count özelliği ile kullanılır ve ilgili alanda kaç kolon seçili olduğu elde edilir. Ayrıca ilgili alandaki belirli kolonlar üzerinde formatting işlemleri yapılabilir. Tabi burda kolondan kastımız tüm bir kolonunun seçimi değil, ilgili alandaki dikey blokların seçimidir. Örneğin A2: B10 alanı içinde Columns(2) seçildiğinde sadece B2: B10 seçilir, tüm B kolonu değil.

**Column**: Tek bir hücre söz konusuysa onun bulunduğu kolonun index numarası, birden çok hücre grubu söz konusuysa ilk hücresinin bulunduğu kolonun index numarası döner.

**Rows** ve **Row** de Columns ve Column'un satır versiyonudur.

Mesela şu tablo;



Sub ZebraYap()

Dim alan As Range

Set alan = Range("B3: D11")

For i = 1 To alan.Rows.Count

If i Mod 2 = 1 Then

alan.Rows(i).Interior.Color = vbBlue

Else

alan.Rows(i).Interior.Color = vbWhite

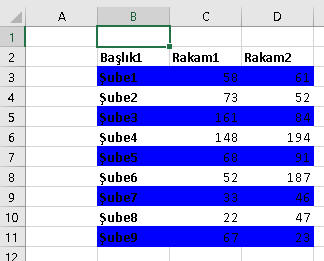
End If

Next i

alan.Columns(1).Font.Bold = True

End Sub

kodu çalıştıktan sonra böyle görünür



Peki bir hücrenin bulunduğu tüm satır veya sütunu seçmek isteseydik? O zaman da **EntireRow** ve **EntireColumn** özelliklerini kullanırız.

ActiveCell.EntireColumn.Select 'aktif hücrenin bulunduğu tüm kolonu seçer

ActiveCell.EntireRow.Select 'aktif hücrenin bulunduğu tüm satırı seçemer

Item özelliğinde olduğu gibi Columns için de range'in sınırları dışındaki index numaraları verilebilir. Bu da bize kolonlarda tek tek ilerlemenin kolay yöntemlerinden birini sunmuş olur. Ör: Range("A1").Columns(2), B2 hücresine ilerlemenin bir yoludur ve döngüsel işlemlerde bir yerden bir yere programatik olarak ilerlememizi sağlar.

#### Satır/Sütun ekleme, silme, gizleme

Makro kaydedicisi ile birkaç işlem yaptım ve kodları aşağı aldım.

Columns("E: E").Select

'Şimdi kolon eklenecek

Selection.Insert Shift: =xlToRight, CopyOrigin: =xlFormatFromLeftOrAbove

Columns("F: F").Select

'Şimdi kolon silinecek

Selection.Delete Shift: =xlToLeft

Rows("4: 4").Select

'Satır eklenecek

Selection.Insert Shift: =xlDown, CopyOrigin: =xlFormatFromLeftOrAbove

'Satır silinecek

Selection.Delete Shift: =xlUp

Columns("E: E").Select

'Gizlemek için bir eylem yerine Gizli özelliğine True değeri veriliyor

Selection.EntireColumn.Hidden = True

Columns("D: F").Select

'Gizli olan kolonu açmak için de Gizli özelliğine False değeri atanıyor

Selection.EntireColumn.Hidden = False

#### Range'i bir diziye atama

Belli bir rangedeki değerleri almanın en hızlı yolu burayı **Variant** türünde bir diziye atamaktır. Ancak tek boyutlu bir dizi değil, satır ve sütunu ifade eden iki boyutlu bir dizi.

Diziler konusunda detaylı göreceğiz, bir dizinin eleman sayısı **Ubound** metodu ile elde edilir. n. boyutundaki eleman sayısı da Ubound(dizi, n) ile. Bu örnekte 1.boyutumuz satırları 2.boyutumuz ise kolonları ifade ediyor.

Sub range\_diziata()

Dim siciller As Variant

siciller = Range(Range("a1"), Range("a1").End(xlDown).End(xlToRight)).Value

'colon sayısı: ubound(siciller,2)

'row sayısı: Ubound(siciller,1) veya kısaca ubound(siciller), 1 defaulat value

Debug.Print UBound(siciller, 2)

Debug.Print UBound(siciller, 1)

'bunu aynen bir yere yapıştırmak için, aynı boyutta olması lazım, bunu da resize ile hallederiz

Range("h1").Resize(UBound(siciller), UBound(siciller, 2)).Value = siciller

End Sub

#### Veri depolama aracı olarak Range

Masaüstü veya Web tabanlı programlamayla uğraşanlar bilirler, aynı oturumdayken programı birkaç kez çalıştırdığımızda önceki çalıştırmalarda elde ettiğimiz bir değeri daha sonra kullanmak için Visible özelliği False olan yani gizli olan bir Textbox'a(veya Labela) bu değeri atarız. Mesela bir tuşa kaç kez basıldığını böyle bir kutu içinde depolayabilir ve 10.kez basıldığında kullanıcıya bir mesaj gösterebilir veya Programdan çıkışı sağlayabilir veya başka bir kodun çalışmasını sağlayabilirsiniz.

İşte Excel'de de Range nesnesini bu amaçla kullanabilirsiniz. Mesela, bir düğmeye ilk kez basıldığında uzun bir kodun çalışmasını, sonraki basışlarda ise daha kısa kodların çalışmasını sağlayabiliriz. Örnek kod aşağıdaki gibi olacaktır. Geçici depolama işini Z1 hücresinde yapacağız(ilk açılışta görünmeyen bir hücre olması nedeniyle. Mümkünse font rengini de beyaz yaparız, veya Z kolonunu gizleriz.)

NOT: Bu yöntem static değişken tanımlamanın bir alternatifidir.

Sub dugme\_click()

'Z1 ilk başta boştur, yani 0'dır.

If Range("Z1").Value2>1 Then

'Sadece son sayfadaki Table'ı refresh et. 10 sn sürer

Else 'Z1=0 ise yani düğmeye ilk kez basılıyorsa

'Tüm sayfalardaki connectionları refresh et. 5 dk sürer.

End If

Range("Z1").Value2=Range("Z1").Value2+1 'Z1dei değeri bir artırıyoruz

End Sub

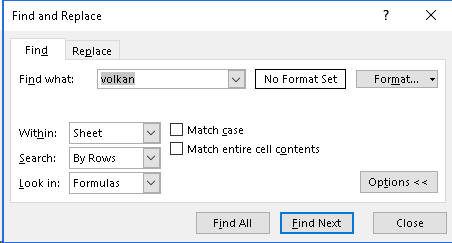
#### Find ve Replace

Excel içindeyken Ctrl+F(veya Home>Find) yaptığımızda karşımıza çıkan **Find** dialog kutusunu ve sonrasında yaptığımız bulma işlemini Range nesnesinin **Find** metodu ile yapıyoruz. Tahmin ettiğiniz üzere bu metod da yine bir range nesnesi döndürür, dönen nesne de aranan şeyin ilk bulunduğu hücredir. Eğer aranan değer bulunmazsa **Nothing** döner.

En basit haliyle aşağıdaki gibi kullanılır.

Dim arabul As Range

Set arabul = Range("A1").CurrentRegion.Find("Volkan")



Gördüğünüz gibi hiç parametre kullanmadık. Böyle bir durumda hangi değerler baz alınır? Şimdi bir genel kültür bilgisi verelim. Siz Excel'de bu araçla çalışırken en son hangi değerleri kullandıysanız bir sonraki aramanızda da bu değerler kullanılır, çünkü arama ayarlarınız kaydedilir. VBA'de de olan budur. Yani parametresiz Find kullanırsanız Excel’de en son kullandığınız arama kriterleri kullanılır. O yüzden tavsiyem bu kriterleri açıkça belirterek yazmanızdır. Aşağıdaki gibi:

Dim arabul As Range

Set arabul = Range("A1").CurrentRegion.Find( \_

What: ="volkan", \_

After: =ActiveCell, \_

LookIn: =xlFormulas, \_

LookAt: =xlPart, \_

SearchOrder: =xlByRows, \_

SearchDirection: =xlNext, \_

MatchCase: =False, \_

SearchFormat: =False)

Parametrelerin açıklamalarını aşağı yukarı tahmin ediyorsunuzdur, zira bunları zaten Excel içinde sık sık kullanıyorsunuz. (Bunlardan bir tek LookAt parametresi kafa karıştırıcı olabilir, çünkü Find dialog kutusundaki Match Case seçeneği için aynen MatchCase parametresi varklen, Match entire cell cotent için MatchEntireContent diye bir parametre beklerken bunun yerine LookAt parametresi var.)

##### Nothing

Dedik ki aradığımız değer bulunmazsa Nothing döndürür, bunu da aşağıdaki gibi sorgulayabiliriz.

If Not arabul Is Nothing Then

arabul.Select

End If

##### Diğer Find detayları

Tüm sayfalarda arama yapmak için For Each döngüsü ile sayfalarda dolaşmak gerekebilir.

Dim ws As Worksheet

Dim arabul As Range

For Each ws In ActiveWorkbook.Sheets

ws.Activate 'veya select

Set arabul = Cells.Find( \_

What: ="volkan", \_

After: =ActiveCell, \_

LookIn: =xlValues, \_

LookAt: =xlPart, \_

SearchOrder: =xlByRows, \_

SearchDirection: =xlNext, \_

MatchCase: =False, \_

SearchFormat: =False)

If Not arabul Is Nothing Then

arabul.Select

Exit For

End If

Next ws

Aradığımız değerin istediğimiz sonucu getirmediğini görür ve aramaya devam etmek istersek **FindNext** metodu kullanılır, bu metod tek bir After parametresi alır.

Cells.FindNext(After: =ActiveCell).Activate

Formatlı arama için **Application.FindFormat** propertyleri kullanılır ve SearchFormat=True yapılır. Record makro ile detaylara bakabilirsiniz.

##### Replace

Replace metodunun kullanımı Find'a benzer. Find'dan farklı olarak Range nesnesi değil Booelan döndürür. Sonuç True ise işlemi yapar, False ise birşey yapmaz. O anda bulunulan hücrenin yeri değişmez.

Cells.Replace \_

What: ="ali", \_

Replacement: ="veli", \_

LookAt: =xlPart, \_

SearchOrder: =xlByRows, \_

MatchCase: =False, \_

SearchFormat: =False, \_

ReplaceFormat: =False

#### Belirli bir hücre grubunun değerini hesaplatma

Application konusunu anlatırken göreceğimiz ve üzerinde epeyce duracağımız konulardan biri da Calculation işlemleridir. Calculate metodu Application nesnesinde var, Worksheet nesnesinde var, Workbook nesnesinde yok ama bir döngü ile sağlanabiliyor. Range'te yok mu? Tabiki var. Diyelim ki, Calculation durumu Manuel set edilmiş durumda ve sayfanızın münferit yerlerinde tonla formül var, siz sadece belirli bir grup hücrede formüllerin hesaplanmasını isteyebilirsiniz. Bunun için küçük bir kod yazıp bunu QuickAccessToolbarınıza koyabilirsiniz. İşte kodumuz.

Sub rangecalc()

Selection.Calculate

End Sub

#### Parent özelliği

Bazen bir hücrenin hangi sayfada olduğunu elde etmek isteriz. Bunun için hiyerarşide bir üst basamağa çıkmamızı sağlayan Parent özelliğini kullanırız.

Debug.Print TypeName(Activecell.Parent) 'Worksheet

Debug.Print Activecell.Parent.Name 'ilgili Worksheetin adı

### Çeşitli Örnekler

#### Bir alanı başlık hariç seçmek

Sıklıkla bir alanı başlık hariç seçmeniz gerekebilecektir. Arkasından bu alanla ne yapmak istiyorsanız yapabilirsiniz. Bunun için aşağıdaki gibi bir fonksiyon tanımlayıp seçili alanı bu fonksiyona parametre olarak gönderebiliriz.

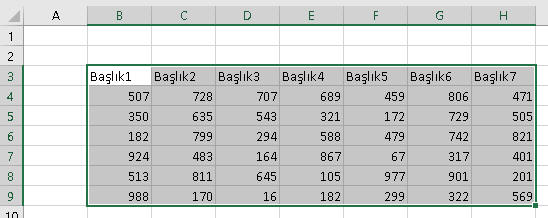
Function başlıkhariçalan(alan As Range) As Range

Dim enalt As Long

enalt = alan.Rows.Count

Set başlıkhariçalan = alan.Resize(enalt - 1).Offset(1)

End Function



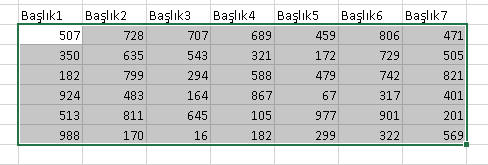
Fonksiyonun kullanımı oldukça basit.

'diyelim ki o anda ilgili alanı seçmişiz

başlıkhariçalan(Selection).Select

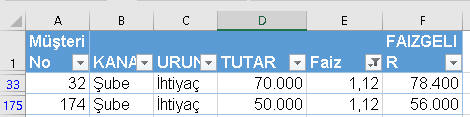
'veya bulunduğumuz hücrenin CurrentRegionunı da gönderebiliriz

başlıkhariçalan(Activecell.CurrentRegion).Select



#### Filtrelenmiş bir alandaki ilk visible(görünür) hücreyi/satırı seçmek

Bazen elinizdeki listeler/tablolar üzerinde filtreleme işlemi yapıp hemen arkasından da filtrelenmiş bu kısmı başlık hariç olarak kullanmanız gerekecektir. Böyle bir durumunda, az önceki gibi başlığın hemen bir altındaki hücreye ilerlememiz yeterli olmayacaktır. Zira görünen ilk hücre aşağıdaki gibi 33.satırda olabilir.



Bu örnekte bizim önce 33.satıra gelebilmemiz lazım. Ondan sonrası kolay: Önce en sağa, ve oradan da en aşağıya kadar seçeriz.

Function ilkvisiblesonrasıalan(alan As Range) As Range

Dim ilk As Range

Dim son As Range

Dim n As Integer

n = alan.Columns.Count

Set ilk = alan.Offset(1, 0).Resize(alan.Rows.Count - 1, 1).SpecialCells(xlCellTypeVisible).Cells(1, 1) 'bu kısım \_

ilk görünen hücreyi verir

Set son = ilk.Offset(0, n - 1)

Set ilktoright = Range(ilk, son)

Set ilkvisiblesonrasıalan = Range(ilktoright, ilktoright.End(xlDown))

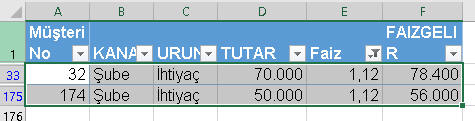
End Function

'kullanım yine oldukça basit

ilkvisiblesonrasıalan(Selection) 'eğer ki hali hazırda alan seçiliyse

ilkvisiblesonrasıalan(Activecell.CurrentRegion) 'alan henüz seçili değilse

ve sonrasında durum şöyledir:



#### Filtre uygulanmış bir alandaki son hücreyi elde etmek

Aşağıdaki gibi filtre uygulanmış bir alandaki görünen son hücreyi elde etmek kolaydır. Ya A1'den aşağı indirip([A1].End(xlDown)) veya sayfanın en altındaki hücreden (1048576. hücreden) yukarı çıkarak. Böylece 13'üncü satırı veya A13 hücresini elde ederiz.

Ancak bazen biz bu örnekten konuşacak olursak 34.satır veya A34 hücresi gerekir. Bunun için bir fonksiyon yazacağız.



Function FiltredekiSonHucre(ws As Worksheet)

On Error GoTo hata

FiltredekiSonHucre = ws.Range(Split(ws.AutoFilter.Range.Address, ": ")(1)).Row

Exit Function

hata:

End Function

Fonksiyonun yaptığı iş basit. F8 ile ilerlerken baktığımızda filtreli alanın adresini aşağıdaki gibi görüyoruz: $A$1: $P$34. Bunu bir metin olarak el alıp ayracı ": " olacak şekilde Split ile bir diziye dönüştürüyoruz. Sonrasında elde edilen dizinin 1.indeksindeki yani 2.elemanını alıyoruz. 1.eleman $A$1 olup ikinci eleman $P$34'tür. Sonra bu stringi Range içine koyup bunu hücre olarak elde edip onun da Row'unu döndürüyoruz.



#### Otomatik mail gönderiminde body'ye Excel hücreleri yapıştırma

Bu olağanüstü faydalı makroyu üstatlardan Ron de Bruin'in sayfasından([http: //www.rondebruin.nl/win/s1/outlook/bmail2.htm](http://www.rondebruin.nl/win/s1/outlook/bmail2.htm)) aldım. Hiç değiştirmeden aynen alıntılıyorum. Siz de aynen bu şekilde kullanabilirsiniz.

Function RangetoHTML(rng As Range)

' Changed by Ron de Bruin 28-Oct-2006

' Working in Office 2000-2016

Dim fso As Object

Dim ts As Object

Dim TempFile As String

Dim TempWB As Workbook

TempFile = Environ$("temp") & "\" & Format(Now, "dd-mm-yy h-mm-ss") & ".htm"

'Copy the range and create a new workbook to past the data in

rng.Copy

Set TempWB = Workbooks.Add(1)

With TempWB.Sheets(1)

.Cells(1).PasteSpecial Paste: =8

.Cells(1).PasteSpecial xlPasteValues, , False, False

.Cells(1).PasteSpecial xlPasteFormats, , False, False

.Cells(1).Select

Application.CutCopyMode = False

On Error Resume Next

.DrawingObjects.Visible = True

.DrawingObjects.Delete

On Error GoTo 0

End With

'Publish the sheet to a htm file

With TempWB.PublishObjects.Add( \_

SourceType: =xlSourceRange, \_

Filename: =TempFile, \_

Sheet: =TempWB.Sheets(1).Name, \_

Source: =TempWB.Sheets(1).UsedRange.Address, \_

HtmlType: =xlHtmlStatic)

.Publish (True)

End With

'Read all data from the htm file into RangetoHTML

Set fso = CreateObject("Scripting.FileSystemObject")

Set ts = fso.GetFile(TempFile).OpenAsTextStream(1, -2)

RangetoHTML = ts.readall

ts.Close

RangetoHTML = Replace(RangetoHTML, "align=center x: publishsource=", \_

"align=left x: publishsource=")

'Close TempWB

TempWB.Close savechanges: =False

'Delete the htm file we used in this function

Kill TempFile

Set ts = Nothing

Set fso = Nothing

Set TempWB = Nothing

End Function

Fonksiyonun kullanımı oldukça basit. Yine aynı sitede örnek kullanım da var. Ancak isterseniz kitabın Outlook otomasyonuyla ilgili sayfalarına da bakabilirsiniz.

## Worksheet

### Giriş

Excel Nesne Modelinde Workbooktan sonraki seviyede sayfalar gelir. Excel’de 4 tip sayfa vardır. Bunlar;

* Çalışma sayfası(**Worksheet**: En çok bunları kullanacağız)
* Chart Sheet(Başlı başına bir sayfa olan grafik sayfaları)
* Macro Sheet(Bildiğimiz makro değil, eskiyle uyumluluk adına duruyor, biz bunlara hiç girmeyeceğiz)
* Dialog Sheet(Eskiyle uyum adına duruyor, bunlara da girmeyeceğiz)

Biz en çok **Sheet** ve **Worksheet** nesnelerini kullanacağız. Yukardaki 4 maddeden anlaşıldığı üzere **Worksheet** nesnesi, **Sheet** nesnesinin bir alt türü oluyor. Sheet nesnesi, Sheets koleksiyonunun, Worksheet nesnesi de hem Worksheets koleksiyonunun hem de Sheets koleksiyonunun bir üyesidir. Bu koleksiyonlar bir dosyadaki tüm sayfaları ifade eder ve genelde döngülerde (Döngü konusunu hiç bilmiyorsanız en azından bir **For Next** için kitabın ilgili kısmına bakıp tekrar gelin), veya toplam sayfa adedini saydırma gibi özelliklerle birlikte kullanılır. Diyelim ki dosyamızda 3 çalışma sayfası bir de grafik sayfası varsa **Sheets.Count** 4 sonucunu döndürürken **Worksheets.Count** 3 sonucunu döndürür.

Bunu kendiniz de deneyebilirsiniz. Yukarıdaki özelliklerde bir dosya hazırlayın ve VBE'de şu kodları çalışıtırın

Debug.Print ActiveWorkbook.Sheets.Count

Debug.Print ActiveWorkbook.Worksheets.Count

**Not**: Biz birçok yerde Sheets ve Worksheets'i birbiri yerine kullanacağız.

### Temel işlemler

#### Sayfalara erişim ve referans

Sayfalar bir koleksiyon üyesi oldukları için onlara koleksiyonun **item** özelliği ve bu özelliğin index numarası ile ulaşabiliriz. Item özelliği koleksiyonların default özelliği olduğu için tüm diğer koleksiyonlarda olduğu gibi bunda da yazılmadan es geçilebilir.

Yani **Worksheets.Item(1)** ile **Worksheets(1)** tamamen özdeştir. Keza Workbooks.Item(1).Sheets.Item(2) ile Workbooks(1).Sheets(2) de özdeştir. Gördüğünüz üzere başka bir Workbooktaki bir sayfaya da Workbook ile sayfa arasına nokta koyarak ulaşıyoruz. Koleksiyonlarda indexler 1'den başlar(En soldaki sayfanın index değeri 1'dir.)

Sayfalara index numarasıyla olduğu gibi sayfa adı ile de, Worksheets("ilksayfa") gibi, ulaşabiliriz.

Mevcutta aktif olan sayfaya da **ActiveSheet** ifadesi ile ulaşırız. Activesheetle ilgili detaylara aşağıda değineceğiz.

#### Seçme ve Aktive etme

Range nesnesinde nasıl bir veya birden çok hücreyi seçmek için **Select** metodu kullanılıyorsa sayfalar için de aynı metod kullanılır. Yine Range'te olduğu gibi tek bir hücreyi aktive etmek için **Activate** metodu kullanılıyordu, burda da yine aynı metod tek bir sayfayı aktive etmek için kullanılır.

Aşağıdaki kod ile tüm sayfalarda dolaşıyor ve her sayfanın ilk hücresine artan bir şekilde sıra numarası yazıyorum.

For Each ws In ActiveWorkbook.Worksheets

ws.Select

Range("A1").Value=i

i=i+1

Next

**Activate** gizli olan sayfaların seçimi için de kullanılabilirken, **Select** ile gizli sayfalar seçilemez. Mesela şimdi bir dosya açın, 5 sayfası olsun, 2.sini gizleyin. Sonra aşağıdaki kodu çalıştırın.

Sub gizlisec\_activateli()

Dim ws As Worksheet ' bu sefer ws'yi Worksheet olarak tanımladım

For Each ws In ActiveWorkbook.Sheets

ws.Activate

Range("a1") = ws.Index

Next

End Sub

Bu kod ile tüm sayfalarda A1 hücresinin dolu olduğunu görürsünüz, 2.sayfayı unhide edip kontrol edebilirsiniz.

Sub gizliyisec1()

Dim ws As Worksheet

For Each ws In ActiveWorkbook.Sheets

ws.Select 'gizli olduğu için hata verir

Range("a1") = ws.Index

Next

End Sub

Bu yukardakini "sayfa visible mı?" diye kontrol ederek de yapalım, bu sefer hata almayız.

Sub gizliyisec2()

Dim ws As Worksheet

For Each ws In ActiveWorkbook.Sheets

If ws.Visible then

Range("a1") = ws.Index

ws.Select

End If

Next

End Sub

#### İsimlendirme

**Name** özelliği ile sayfanın ismini elde eder veya onu değiştiririz. Yani hem okunan hem yazılan(sets and returns) bir özelliktir. Index ise değiştirilemez bir özellik olup sadece okunurdur.

ws.Name="Yeni sayfa"

#### Yeni sayfa açma(ekleme) ve silme

Yeni sayfa açmak/eklemek için **Add** metodu kullanılır.

**Syntax: Worksheets.Add(Konum)** Konum belirtmezsek aktif sayfanın soluna ekler. Konum belirtmek için parametreleriyle birlikte kullanırız. **Worksheets.Add Before: =Worksheets("Sheet3")**

Sayfayı yaratırken aynı anda ismini de verebiliriz.

Worksheets.Add(Before: =Worksheets("Sheet2")).Name = "Yeni Sayfa"

Sayfaları silmek için **Delete** metodunu kullanırız.

Worksheets(1).Delete

Silme işlemlerinde Excel bize uyarı çıkarır, silmek istediğimizden emin miyiz diye. Bu tür uyarılar bazen uzun makrolarda sorun çıkarabilir, hele bir de bilgisayarımızın başında değilkenki bir saate ayarlanmış bir kod ise biz gelip müdahale edene kadar ekranın takılı kalmasına neden olur, ve varsa sonrasında ayarlanmış kodların da çalışmasını engellemiş our. Bu tür durumlarla karşılaşmamak için kodların başına **Application.DisplayAlerts=False** cümleciği yazılır, ilgili kodlar bittikten sonra da True'ya çevrilir. Bunun detaylarına Application konusunda tekrar geleceğiz.

#### Sayfaları Gizleme ve Gizleneni tekrar gösterme

Sayfaları gizlemek veya tekrar göstermek için beklenenin aksine bir eylem değil **Visible** adında bir özellik kullanıyoruz. Bu özellik booelan tipinde değer döndürür.

Worksheets("Sheet2").Visible = False/True

Bir diğer yöntem de bu özelliğe True/False atamak yerine **XlSheetVisibility** enumerationlarını kullanmak olabilir. Alacağı değerle şöyledir.

**xlSheetVisible**: Sayfayı gösterir

**xlSheetHidden**: Sayfayı gizler

**xlSheetVeryHidden**: Sayfayı öyle bir gizler ki, kullanıcılar sayfalara sağ tıklayıp Unhide tuşuna bastıklarında bile gizli sayfa listesinde görünmez. Bunu nerede kullanmak isteyebilirsiniz? Mesela workbook/worksheet koruma şifrenizi veya bir veritabanı bağlantısı için yazdığınız Connection String içindeki bağlantı parolasını VBA kodu içine doğrudan yazmak istemiyorsunuzdur. Bunu bir xlVeryHidden nitelikli sayfanın bir hücresine yazıp buradan okutturabilirsiniz.

Bu konuyla ilgili biraz aşağıda örneğimiz olacak.

#### Taşıma ve Kopyalama

**Move** ve **Copy** metodularını kullanırız. İkisi de Before(Önce) ve After(Sonra) olmak üzere iki parametre alır.

Worksheets("Sheet3").Move After: =Worksheets("Sheet1")

Worksheets("Sheet3").Copy Before: =Worksheets("Sheet1")

Worksheets("Sheet1").Copy Before: =Workbooks("ExcelVBA.xlsm").Sheets("Sheet3")

Worksheets("Sheet1").Copy 'yeni bir dosya açıp ve direkt bu dosyaya kopayalar

Bu metodları tek başına kullandığınızda yeni bir dosya açar ve oraya taşır/kopyalar. Aşağıdaki bir örnekte bu özelliği kullanıyoruz.

Birkaç kopyalama ve taşıma örneğini macro recorder ile kendiniz de yapabilirsiniz. Basit bir konu olduğu için daha fazla detaya gerek görmedim.

#### Sayfalarda koruma (Protection & Unprotection)

Yazdığımız kodlar, Sayfa korumalı bir dosyada işlem yapmaya kalkarsa şöyle bir hata alırız: **"Run-time error '1004': Application-defined or object-defined error"**.

Bu sorunun üstesinden gelmek için, öncesinde sayfanın korumalı olup olmadığını kontrol edebilir, varsa korumayı kaldırabiliriz. Ama bunun da bir sakıncası olabilir, o da şu ki, kodlarda ilerlerken başka bir hata çıkarsa program durur ve sayfamız korumasız kalır. Bunun için bir de Hata Yakalama kodu yazmamız ve ilgili yerde tekrar koruma koymamız gerekir.

Korumaya almak **Protect**, korumayı kaldırmak **Unprotect** metodu ile sağlanır.

Şimdi bir örnek yapalım. Yeni bir dosya açın ve ilk sayfasına koruma uygulayın, şifresi de 1234 olsun. Sonra da şu kodu ekleyip çalıştırın.

Sub sayfakoruma1()

Dim ws As Worksheet

Set ws = ActiveSheet

If ws.ProtectContents = True Then 'koruma var mı diye sorguluyoruz

ws.Unprotect Password: ="1234"

End If

'On Error GoTo hatayakala 'herhangi bir hatada sayfayı tekrar korumaya almak için ilgili yönlendirmeyi yapıyoruz

Range("A1") = Environ("USERNAME")

'çeşitli işlemler

'0'a bölme olacak ve hata alacak

a = InputBox("Bir sayı girin")

b = InputBox("Bu sayıyı kaça bölelim")

MsgBox "Sonuç: " & a / b

ws.Protect Password: ="1234"

Exit Sub

'hatayakala:

'ws.Protect Password: ="1234"

End Sub

İlk olarak, a için abc değerini girin, b için bir sayı girin. İkinci denemede de b için 0 değerini girin. İlki sayısal bir değer girmediğimiz için, ikincisi de 0'a bölmeye çalıştığı için hataya neden olacaktır. Hata yakalama kodları commentli olduğu için de dosyamız protectionsız kalacaktır.

Şimdi yukardaki commentli kısımları commentsiz hale getirip tekrar çalıştıralım. Bu sefer hata yakalanacak ve çıkmadan önce tekrar şifreleme yapılacaktır.

**NOT**: Filtrelemeye izin verme gibi seçeneklerden faydalanmak için makro kaydetme aracından faydalanabilirsiniz. Ancak bu şekilde protection koymaya çalıştığınızda recorder şifreyi kaydetmez, bunu manuel eklemeniz gerekir.

### İleri Seviye İşlemler

#### Protectionlı sayfalara devam

Yukarda korumalı sayfalarda çalışma şeklini görmüştük. Korumalı sayfalarla çalışmanın daha şık bir yolu var aslında. **UserInterFaceOnly** parametresi.

Bu parametreye True değeri atanırsa, sayfa sadece kullanıcı işlemlerine karşı korumalı olur, VBA kodları için koruma geçersiz olur. Bu parametrenin default değeri False olup değer atanmaması durumunda tahmin edeceğiniz üzere full koruma sağlanır.

Yanlız unutulmamalıdır ki, bu argüman sadece bir kereliğine koruma sağlar. Dosyayı kapatıp tekrar girdiğinizde, full korumalı açılır. Bu özelliğin daimi olması için Workbook\_Open eventi içine uygun kod yazılabilir.

Private Sub Workbook\_Open()

Dim ws As Worksheet

For Each ws in Activeworkbook.Worksheets

ws.Protect Password: ="1234",UserInterFaceOnly: =True

Next ws

End Sub

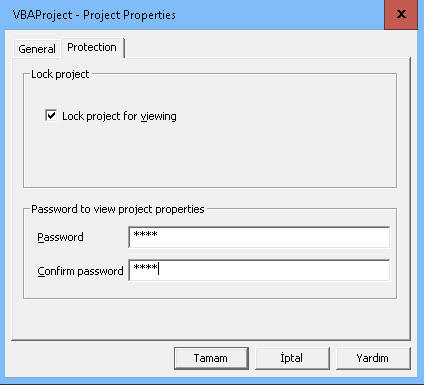
Bu arada olur da sayfanın korumasının ne şekilde yapıldığını görmek isterseniz ProtectionMode özelliğine bakmanız gerekir. True ise UserInterFaceOnly moduyla korunmuştur, false ise full koruma vardır.

Akabinde kodla sayfa içinde bir yerlere bir şeyler yazdırabilirsiniz. Mesela bir veritabanından bir veriyi refresh edebilir, refresh işleminin tarih ve saatini bir hücreye yazdırabilirsiniz.

Bu özelliğin ne için kullanıldığını hala gözünde canlandıramamış olanlar için şöyle belirtelim: Bunun daha zahmetli alternatifi şudur: Korumayı geçici süre kaldırmak, gerekli hücrelere verileri yazdırmak, sonra korumayı geri koymak olurdu. Zahmetli olduğu kadar risklidir de, zira korumayı geri aktive etmeyi hatırlayacağınızın bir garantisi yok.

#### Şifrenin VBE'de görünmemesi

Korumalı bir sayfada korumayı koyma/kaldırma işlemlerini VBA kodları şeklinde yapıyorsanız, meraklı bir kullanıcı, birazcık VBE ortamını da biliyorsa, bir şeklide VBE'ye girip kodları karıştırırsa şifreleri görebilir. Zira Protect ifadesinden sonraki Password: =""" parametresi ayan beyan görünmektedir. Bunun için VBE üzerinden kodlarınızın görünmemesi için bir şifre koymanız gerekebilir.



Ancak diyelim ki bir nedenle VBA protection koymak istemiyorsunuz, Workbook Protection da koymak istemiyorsunuz, bu durumda diğer alternatif de şu olabilir: Şifreyi bir sayfanın A1(veya istediğiniz herhangi bir) hücresine koyup, gereken yerlerde şifreyi buradan okumaktır. Sonra bu sayfayı da **xlVeryHidden** olarak saklamak, böylece kodları kurcalayan kişi "ya burda Sheets(1)'nin A1 hücresinden şifreyi alınıyor görünüyor, ama A1 hücresi boş, bu ne biçim iş anlamadım ben" diye yakınıp durur. Zira deneyimli bir makro kullanıcısı olmadığını varsayarsak kendisinin xlVeryHidden sayfa türünden haberi olmayacaktır. "Gizli bir sayfa mı acaba?" deyip sayfa sekmelerine sağ tıklayıp Unhide dediklerinde de bu sayfayı orada göremeyecekler ve çıldırıp duracaklardır.

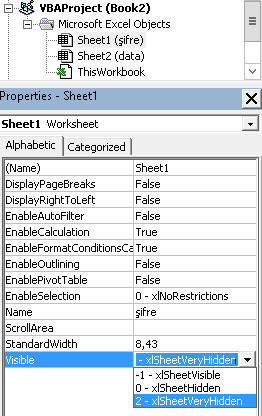
Bunun için ya şu kodu yazacağız,

Sub şifreleme()

Sheets(1).Visible = xlVeryHidden

End Sub

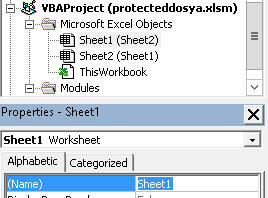
Ya da VBE'de Project penceresinden ilgili sayfayı seçip Properties penceresinden de şu ayarı yapacağız:



Kod yazarak yapma yolunu seçersek bunu hemen silelim ki bir başkası böyle bir sayfanın varlığından haberdar olmasın. Tabi umarız ki kullanıcı, bu Project penceresindeki şifre sayfasını görüp de Excel önyüzünde göremeyince işkillenmez.

Şimdi bir örnekte gösterelim.

* Yeni bir dosya açın
* 2.sayfa yoksa yaratın ve bunu **xlVeryHidden** olarak ayarlayın
* 2.sayfann A1 hücresine bir şifre yazın
* Şifre yazdığınız hücreye Name olarak şifre adını verin, formatını beyaz renk yapın görünmez olsun, ayrıca **Format Cells>Protection** menüsünden Hidden diye de işaretleyin
* İlk sayfaya Developer menüsünde bir düğme ekleyin
* Düğmenin click eventine aşağıdaki kodu yazalım
* Sayfaların kod isimlerini tersine çevirin ki iyice kafa karıştırsın. (1.sayfanın adını Sheet2, 2.sayfanın adını Sheet1 yapıyoruz)



Sub Button1\_Click()

If ActiveSheet.ProtectContents = True Then

ActiveSheet.Unprotect [şifre] 'Name'e az kullanılan bu yöntemle ulaşarak daha fazla kafa karıştırıyorum

ActiveSheet.Shapes("Button 1").OLEFormat.Object.Text = "Koruma kaldırıldı"

Sheet1.Unprotect [şifre] 'şifre sayfasına code adıyla ulaşıp daha da kafa karıştırıyoruz ki kurcalayan kişi artık vazgeçsin

Else

ActiveSheet.Shapes("Button 1").OLEFormat.Object.Text = "Korumaya alındı"

ActiveSheet.Protect [şifre]

Sheet1.Protect [şifre]

End If

End Sub

Bu yöntemi, kırması zor olsun diye çok uzun bir şifre belirlediğinizde de kullanabilirsiniz. Uzun şifreyi xlVeryHidden sayfaya yazarsınız. Ana sayfada da bir hücre belirlersiniz, oraya çift tıklandığında dosya Unprotect olur, böylece uzun şifreyi manuel girmek zorunda kalmazsınız.

Bir diğer alternatif de Workbook koruması koymaktır. Şifreyi de yine herhangi bir gizli sayfaya koyabilirsiniz, bu sefer VeryHidden olmasına gerek yok, zaten sayfaları hide/unhide özelliği tamamen pasif olacak.

Bu yöntemler ayrıca, veritabanı bağlantılarının şifrelerini almak için de kullanılabilir, ancak ribbonun da pasifleştirilmesi lazım, yoksa ribbondaki Properties'ten de girip şifreyi görebilirler. Veya datayı çektikten sora ilgili Table'ı Range'e döndümeniz gerekir ki şifre görünmesin. Bunları, yeri geldiğinde detaylıca göreceğiz.

##### ****ActiveSheet****

Bir sayfanın özelliğiyle ilgili bir ifade yazıyorsak ve bu özelliğin önünde sayfa ismi yoksa, Excel bunun o anda aktif olan sayfanın bir özelliği olduğunu anlar. Bu bağlamda ActiveSheet.Range("A1") ile Range("A1") tamamen aynı şey olarak algılanır.

Ancak böyle Worksheet ifadesini belirtmeden kod yazmanın bir istisnası var. Eğer, makronuz süreç içinde Sayfa modüllerinden birine girerse, veya direkt oraya yazdığınız bir makroyu çalıştırıyorsanız ve o anda başka bir sayfadaysanız, kodlar işlemi o aktif sayfada değil, ilgili sayfa modülünün sayfasında yapar. Örneğin Sheet1, Sheet2, Sheet3 şeklinde 3 sayfanız olsun. Sheet1 modül sayfasında şu kodu yazalım;

Sub deneme()

Range("a1") = 2

End Sub

Şimdi 2.sayfayı aktif hale getirelim ve VBE'ye geçip Sheet1 modülündeki kodu çalıştıralım.

Gördüğünüz gibi o an aktif sayfa 2.sayfa olmasına rağmen 1.sayfanın A1 hücresine 2 değeri yazıldı.

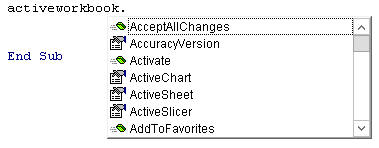
O yüzden sadece Workbook modülü ve standart modüllerdeki kodlarda sayfa ismi belirtilmediyse Activesheet olarak algılanma söz konusudur diyebiliriz.

##### Activesheet ve Dönen Değer

Aktif nesnelerle ilgili önemli bir konu da, nesnenin dönen değerine göre intellisense'in çalışıp çalışmayacağıdır. Eğer nesnenin dönen değeri bir Obje tipinde ise intellisense çalışmaz, obje tipinde değil de spesifik bir nesne tipinde ise intellisense çalışır. Mesela Activeworkbookun dönen değeri bir Workbook olup, hemen arkasından bir nokta(.) yazınca intellisense çalışır ve bu nesneye ait özellik ve metodlar listelenir.



Bu da ActiveWorkbookun intellisense görüntüsü:



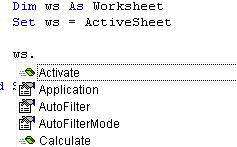
Activesheet'in ise dönen değeri Object’dir, bunu daha önce de görmüştük. Zira activesheet normal bir çalışma sayfası(worksheet) da olabilir, bir grafik sayfası da. Doğal olarak bunların özellik ve metodları da farklıdır. Birden fazla dönüş değeri olan nesneler de Object tipli olmaktadırlar. O yüzden bunlarda intellisense çalışmaz.



Activesheetin intellisense'i de bu nedenle çıkmamaktadır:



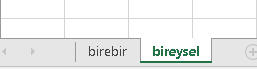
Peki Activesheet'e ait intellisensi çıkarmanın hiç mi yolu yok? Tabi ki var: Onu bir Worksheet nesnesine atamak.



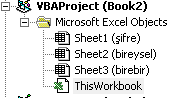
#### ****Sayfal********arın kod ismi**** ile gerçek ismi

Sayfalar yaratıldıkları anda "Sheet 1", "Sheet 2" gibi isimlere sahiptirler. Kullanıcı isterse bunların isimlerini ve sırasını sonradan değiştirebilir.

Sayfaların bir de, yaratıldıkları anda sahip oldukları Sheet1(bu sefer bitişik yazılan), Sheet2 gibi kod adları vardır, bunlar da VBE'deki properties penceresinden yeniden adlandırılabilir, ama pratikte daha çok sayfaların Excel arayüzündeki gerçek isimlerinin yeniden adlandırıldığı görülür. Sayfaların Excel’deki sırası değişse bile kod isimlerinin ismi değişmediği için sırası aynı kalır. Aşağıdaki görüntülerde bunu görebiliyoruz. İlk sayfa gizlenmiş, ikinci ve üçüncü sayfa ise yer değiştirmiş.



Ancak gördüğünüz gibi kod isimleri aynı duruyor.(Tabiki kod isimleri de değişirse bunlar da alfabetik olarak sıralanacaktır)



İşte sayfalara ulaşmanın bir şeklini daha görmüş olduk, kod isimlerle. Ancak dikkat edilmesi gereken bir nokta var, o da kod isimlerin sadece  ilgili dosya içinden(normal modül veya thisworkbook ile sheet modülleri) ulaşılabilir olduğudur. Şimdi tüm yöntemleri bir arada görelim;

Sub sayfalara\_erişim()

Sheet1.Select 'kod isimle. (sadece bunda intellisense çıkar)

Sheets(1).Select 'Sheets koleksiyonu ve index

Worksheets(1).Select 'Worksheets koleksiyonu ve index

Sheets("bireysel").Select 'Sheets koleksiyonu ve sayfa adı

Worksheets("bireysel").Select 'Worksheets koleksiyonu ve sayfa adı

End Sub

### Filtre ve Sıralama işlemleri

Sıralama ve Filtreleme işlemleri her ne kadar hücreler üzerinde yapılıyor olsa da Worksheet nesnesinin özellikleridirler. Öyle çok kompleks bir tarafları olmadığı için Macro Recorder ile kaydedilmiş bir kod üzerinden inceleyebiliriz.

ActiveWorkbook.Worksheets("calculatedlar").Sort.SortFields.Clear

ActiveWorkbook.Worksheets("calculatedlar").Sort.SortFields.Add Key: =Range( \_

"F2"), SortOn: =xlSortOnValues, Order: =xlDescending, DataOption: = \_

xlSortNormal

With ActiveWorkbook.Worksheets("calculatedlar").Sort

.SetRange Range("A2: F175")

.Header = xlNo

.MatchCase = False

.Orientation = xlTopToBottom

.SortMethod = xlPinYin

.Apply

End With

Recorder'ın ürettiği kodda bir iki yeri değiştirmek gerekir, böylece makromuz daha dinamik hale gelir.

* Worksheets("calculatedlar")>ActiveSheet yapalım
* Key: =Range("F2")>Key: =ActiveCell(veya duruma göre kalabilir)
* Range("A2: F175")>Bunu nasıl değiştireceğimizi Range konusunda görmüştük

Keza filtreleme işleminin kodu da basit olup, dinamik hale getirilecek kısımları değiştirmek yeterlidir.

Range("B1").Select

ActiveSheet.Range("$A$1: $F$175").AutoFilter Field: =2, Criteria1: ="Şube"

Bu örnekte Range("$A$1: $F$175") yerine Range("A1").CurrentRegion denilebilir. Gördüğünüz gibi Sort işleminde başlık hariç tutulurken burada başlık satırı dahildir. O yüzden Resize ve Offset kullanmadan işimizi halledebiliriz.

#### Filtre modları ve filtreyi kaldırma

Filtrelerle ilgili olarak kafa karışıklığına neden olabilecek iki konu var. Filtrenin açık/kapalı olması ile kriterin uygulanıp/uygulanmamış olması. Bunlardan ilki **AutoFilterMode** özelliği ile elde edilirken ikincisi **FilterMode** özelliği ile elde edilir.

Filtreyi uygulamak/kaldırmak Range nesnesinin **AutoFilter** metodu ile olur.

* Filtre henüz yokken kullanılırsa filtre uygulanır
* Yanında parametre yoksa sadece filtre okları görünür
* Parametreyle uygulanırsa direkt ilgili filtreleme yapılmış olur
* Filtre varken kullanılırsa kriterler silinir ve filtre okları kalkar

Kriter uygulanmış bir data kümesinde kriterleri silmek ama filtre oklarını açık bırakmak için Worksheet nesnesinin **ShowAllData** metodu kullanılır.

Aşağıda bütün bu durumları anlatan güzel bir örnek var. Bu örnekte sayfadaki data kümesinin rasgele olarak 4 durumundan birine girmesini sağlıyoruz, sonra hangi durumdaysa onunla ilgili bir mesaj veriyorum.

Sub filtresub()

a = WorksheetFunction.RandBetween(1, 4)

MsgBox "Case " & a & " gerçekleşecek"

Select Case a

Case 1 'o an kapalıysa kapalı kalmaya devam, açıksa kapanır

ActiveSheet.AutoFilterMode = False 'buna sadece false atanıyor, true atanamaz

Case 2 'açıkken filtre konursa kapanır

If ActiveSheet.AutoFilterMode = True Then Selection.AutoFilter

Case 3 'kapalıyken filtre konursa açılır

If ActiveSheet.AutoFilterMode = False Then Selection.AutoFilter

Case 4 'filtre okları aktif olsa da olmasa da burası çalışır

ActiveSheet.Range("$A$1").CurrentRegion.AutoFilter Field: =2, Criteria1: =Range("b2")

End Select

If ActiveSheet.AutoFilterMode = True Then 'filtre okları açıksa. Case 3 veya 4

If ActiveSheet.FilterMode = False Then

MsgBox "Case 3: Filtre açık ama kriter yok"

Else

MsgBox "Case 4: filtre açık ve kriter var, şimdi kriter kaldırılacak, ama filtre açık kalacak"

ActiveSheet.ShowAllData

End If

Else 'filtre okları yoksa, yani henüz bir autofilter düğmesine basılmamışsa

MsgBox "Case 1 veya 2: filtre yok"

End If

End Sub

### Diğer Önemli İşlemler

#### Sayfa ayarları

Page Setup, Print v.b işlemleri için macro recorderdan faydalanmanızı öneriyorum.

#### Calculation

Calculation konusuna Application nesnesine detaylıca değiniyoruz. Önemli bir metod olup Application konusunda mutlaka incelemenizi tavsiye ediyorum.

#### Sayfalarda gezinme

Sayfalarda dolaşma işini yukarıdaki birçok örnekte gördüğümüz gibi bir döngü aracılığı ile yapabilmekle birlikte, sadece bir sayfa geri veya ileri gitme işini, sayfa adı v.s yazmadan kolayca yapmamızı sağlayan iki özellik var.

Activesheet.Next.Select ' sonraki sayfa

ActiveSheet.Previous.Select ' önceki sayfa

#### Hafızadan birşey yapıştırma

Hafızadaki(Clipboard) bilgiyi aktif sayfadaki aktif hücreye yapıştırmak için iki metod var. Birincisi Worksheet sınıfının normal **Paste** metdou, ikincisi Range nesnesinin **PasteSpecial** metodu. **Worksheet.Paste** metodu ile Excel arayüzünde yaptığımız gibi içerikte ne varsa yapıştırılır: Veri, formül, format v.s. **Range.PasteSpecial** ile ise yine Excel arayüzde yaptığımız özel yapıştırma türlerini yapabiliyoruz. Sadece değerleri, sadece formatı v.s.

Aşağıdaki kod ile 1.sayfada aktif hücrenin CurrentRegion'ındaki hücreleri 2.sayfadaki aktif hücreye yapıştırıyoruz.

Sheets(1).Select

ActiveCell.CurrentRegion.Select

Selection.Copy

Sheets(2).Select

ActiveSheet.Paste

Bu arada Worksheet sınıfının da PasteSpecial metodu var ama biz onu çok kullanmayacağız, zira bununla Grafik gibi nesneleri veya Access gibi diğer uygulamalardan birşeyler yapıştırabiliyorsunuz.

ActiveChart.ChartArea.Copy

Range("M2").Select

ActiveSheet.PasteSpecial Format: ="Picture (PNG)", Link: =False, \_

DisplayAsIcon: =False

Aşağıdaki örneği de [MSDN](https://msdn.microsoft.com/en-us/library/office/ff835858.aspx)'den aldım, burada diğer format seçeneklerine de bakabilirsiniz. Başka programlarla çalışmayacaksanız çok sık ihtiyacınız olacağını sanmam.

Worksheets("Sheet1").Range("F5").PasteSpecial \_

Format: ="Picture (Enhanced Metafile)", Link: =False,

DisplayAsIcon: =False

#### Parent özelliği

Bazen bir sayfanın hangi workbookta olduğunu elde etmek isteriz. Bunun için hiyerarşide bir üst basamağa çıkmamızı sağlayan Parent özelliğini kullanırız.

Debug.Print TypeName(Activesheet.Parent) 'Workbook

Debug.Print Activesheet.Parent.Name 'ilgili Workbook'un adı

### Çeşitli Örnekler

#### Tüm sayfaları ayrı workbooklar olarak kaydetme

Bu örnekte açık olan bir dosyadaki tüm sayfaları ayrı ayrı dosyalar olarak seçilen bir klasöre kaydediyoruz.

Sub sayfaları\_wb\_olarak\_kaydet()

Dim ws As Worksheet

Dim wb As Workbook

Dim fd As FileDialog

Dim klasör As String

Dim dosya As String

On Error GoTo hata

Application.ScreenUpdating = False

Set wb = Application.ThisWorkbook

Set fd = Application.FileDialog(msoFileDialogFolderPicker)

If fd.Show = True Then

klasör = fd.SelectedItems(1)

Else

Exit Sub

End If

For Each ws In wb.Worksheets

ws.Copy

'Burada bir versiyon kontrolü yapılmasında fayda var, ama örneği basitleştirmek adına onu es geçiyorum ve default formatta kaydetmesine izin veriyorum

dosya = klasör & "\" & Application.ActiveWorkbook.Sheets(1).Name & ".xlsx"

ActiveWorkbook.SaveAs dosya

ActiveWorkbook.Close False

Next

Call Shell("explorer.exe " & klasör, vbNormalFocus)

Application.ScreenUpdating = True

Exit Sub

hata:

Application.ScreenUpdating = True

MsgBox Err.Description

End Sub

#### Tüm sayfaları Unhide etme

Bu örnekte tüm gizli sayfaları açıyoruz. Buna QuickAccessBardan erişmek isteyebilirsiniz, ben öyle yapıyorum açıkçası.

Sub tümsayfalarunhide()

For i = 1 To Sheets.Count

Sheets(i).visible = True

Next i

Sheets(1).Select

End Sub

#### İlk sayfa hariç tümünü hide etme

Diyelim ki bir sayfasında Karne, diğer sayfalarında bu karneyi besleyen sayfaların olduğu bir dosyanız var. Sık sık sayfaları Unhide ve tekrar Hide etme ihtiyacınız oluyor. Bir üstteki kod ile tüm sayfaları Unhide etmiştik. Datayla oynadıktan sonra şimdi tekrar gizleyeceğiz. Bunu da QuickAccessBara eklerseniz müthiş pratiklik sağlar. Bu kodu Karne sayfasındayken çalıştırmanız gerekiyor.

Sub aktifhariç\_hideall()

Dim ws As Worksheet

Set ws = ActiveSheet

For i = 1 To Sheets.Count

If i <> ws.Index Then

Sheets(i).visible = False

End If

Next i

End Sub

#### Tüm sayfalarda ilk kolonda A'dan Z'ye sıralama

Diyelim ki size birçok sayfası olan bir dosya geldi. Ama sayfaların hiçbiri sıralı değil. Bu örnekte tüm sayfalarda A'dan Z'ye sıralama işlemini tek seferde yapıyoruz

Sub tümsayfalarda\_sırala()

For i = 1 To Sheets.Count

Sheets(i).Select

ActiveWorkbook.Sheets(i).Sort.SortFields.Clear

ActiveWorkbook.Sheets(i).Sort.SortFields.Add Key: =Range("A1"), \_

SortOn: =xlSortOnValues, Order: =xlAscending, DataOption: =xlSortNormal

With ActiveWorkbook.Sheets(i).Sort

.SetRange Range(Range("A2"), Range("a2").End(xlDown).End(xlToRight))

.header = xlNo

.MatchCase = False

.Orientation = xlTopToBottom

.SortMethod = xlPinYin

.Apply

End With

Next i

End Sub

|  |  |  |
| --- | --- | --- |
|  |  |  |

## Workbook

### Giriş

Excel uygulamasında o anda açık olan tüm dosyalar Workbooks koleksiyonunu oluşturur ve **Workbook** nesnesi de bu **Workbooks** koleksiyonunun bir üyesidir. Bu kitap boyunca bazen Workbook ifadesini bazen dosya ifadesini kullanacağım.

Bu arada bir de **Window** nesnesi vardır ki, çoğu durumda Workbook nesnesi ile özdeş kullanılabilir. Window'un farkı şudur, bir dosya açtığınızda bu tek bir workbooktur, ancak bunu birkaç pencere şeklinde gösterebilirsiniz, işte bunların her biri Window nesnesidir. Bu nesneye de yer yer değinip onun özelliklerini de inceleyeceğiz. Bu window(pencere) konusunu tam bilmiyorsanız Google'da Excel Window araması yapabilir, ek bilgi alabilirsiniz.

### Temeller

#### Erişim

Workbooklar bir koleksiyon üyesi oldukları için onlara koleksiyonun **Item** özelliği ve bu özelliğin index numarası ile ulaşabiliriz. Item özelliği koleksiyonların default özelliği olduğu için tüm diğer koleksiyonlarda olduğu gibi bunda da yazılmadan es geçilebilir. Yani **Workbooks.Items(1)** ile **Workbooks(1)** tamamen özdeştir. Koleksiyonlarda indexler 1'den başlar, ilk açılan dosyanın indexi 1'dir.

Workbooklara index numarasıyla olduğu gibi dosya adı ile de, **Workbooks("Bütçe2016.xlsx")** gibi, ulaşabiliriz. Dosya ismini Window nesnesi ile de kullanabiliriz. Windows("Bütçe2016.xlsx") gibi.

#### Mevcut bir Workbook'u Açma

Diskte kayıtlı olan bir dosyayı **Open** metodu ile açarız. Hemen makro kaydedici ile bir dosya açıyorum ve koduma bakıyorum.

Sub Macro3()

Workbooks.Open Filename: ="C: \Users\Volkan\Desktop\deneme.xlsx

End Sub

NOT: Bunu yazarken Filename ifadesini kullanmadan veya parantez içinde şöyle de yazabilirdim: **Workbooks.Open ("C: \Users\Volkan\Desktop\a.xlsx").** Parametre yazımının detayları için İleri **Temeller>Terminoloji** konusuna bakınız.

Open metodunun birçok parametresi var ama ben burada sadece önemli olduğunu düşündüğüm 2 parametreden bahsetmek istiyorum. Bunlardan ilki **ReadOnly** parametresi. Şimdi diyelim ki, kullanıcılar için UserForm kullanarak bir arayüz veya bir Add-in oluşturdunuz. Oradaki bir buton da bir dosyayı açacak, ama dosyada ara ara güncellemeler yapmanız gerekiyor, hatta dosyayı schedule programına aldınız ve gece belirli bir saatte açılıp refresh olacak ve kaydedilecek. Bunun için dosyanın kimsede açık olmaması lazım. O yüzden kullanıcılar bu butona tıkladıklarında bunun Readonly açılmasını sağlamak akıllıca olurdu. İşte bu parametre ile bunu yapabiliyoruz, tabi buna True değerini atayarak.

Peki, diyeceksiniz ki, "Bizim bölümdeki bazı kullanıcılar çok cin veya umarsız, onlara hazırladığım arayüzü/add-ini değil de doğrudan dosyanın bulunduğu klasörden dosyayı açabilirler". Bunun da çözümü var, Dosyanın Workbook\_Open event modülüne dosyanın Readonly açılıp açılmadığını kontrol eden bir kod yazarsınız. (Bu detayları yeri geldikçe göreceğiz). Kodlar şöyle olacaktır:

'dosyayı açmaya yarayan kod

Sub dosyaac()

Workbooks.Open "C: \Users\Volkan\Desktop\deneme.xlsx", ReadOnly: =True

End Sub

'----------------------

'bu dosyanın Workbook\_Open modülü

Sub Workbook\_Open()

'ilk başta açan kullanıcının sizin dışında bir kullanıcı olup olmadığı kontrolü yapaım

If Environ$("computername") = "bilgisayar adınız" then Exit Sub 'veya username ile de kontrol edilebilir

If Me.Readonly=False Then

MsgBox "Lütfen Add-in üzerinden giriş yapınız"

Me.Close SaveChanges: =False

End If

End Sub

İkinci önemli parametre de Updatelinks'tir. Bu parametre ile, dosya açıldığında içinde başka workbooklara link varsa bunların güncellenip güncellenmeyeceğini belirtmiş oluruz. Eğer bu parametre belirtilmediyse Excel bize sorar. **DisplayAlerts=False** kullanımının baskılayamadığı tek uyarı şekli budur, o yüzden bunu kendi içinde halletmemiz gerekir. **0**, güncelleme yapılmasın; **3**, yapılsın demektir.

Linklerle ilgili bir başka önemli metod da LinkSources metodu olup, bir dosyanın başka bir dosyadan(veya MS uygulamasından) link alıp almadığını gösterir. Bu metodun dönüş değeri dizidir. Tüm linkleri bir dizi içinde ayrı elemanlar olarak depolar. Eğer link yoksa dizi boştur ve IsEmpty ile yakalanabilir.

linkler = ActiveWorkbook.LinkSources(xlExcelLinks)

If Not IsEmpty(linkler) Then

For i = 1 To UBound(aLinks)

MsgBox "Link " & i & ": " & Chr(13) & linkler(i)

Next i

Else

MsgBox "Dosya başka dosyalara link içermemektedir."

End If

##### Auto\_open makrosu

Workbook\_Open event handlerına benzer bir de Auto\_open prosedürü vardır. Bu, daha çok eskiden kullanılırdı, ancak Excel 2000'den sonra Workbook\_Open event makrosu devreye girdiği için buna pek gerek kalmadı, eski makroları destek adına yaşamaya devam ediyor. O yüzden olur da elinizde sizden önce birilerinin yazdığı bir Auto\_open kodu varsa ve bunu değiştirmek istemiyorsanız aşağıdaki bilgi önemli olacaktır.

Biz henüz eventlere(olaylara) gelmedik, neden bundan bahsediyorum. İşte, VBA içinden çalışan bir Open metodu Auto\_open makrosunu tetiklemez, bu makro sadece dosya manuel olarak yani Excel içinden veya Windows Explorerdan açıldığında tetiklenir. Bunun için ayrıca dosyayı açtıktan sonra bir de şu kodu eklemek lazım.

Workbooks.Open "C: \Users\Volkan\Desktop\deneme.xlsx"

ActiveWorkbook.RunAutoMacros xlAutoOpen

Bu arada Workbook\_Open makrosunda dosyanın nasıl açıldığı önemli değildir. Manuel de açılmış olsa, kod ile de açılmış olsa tetiklenir.

#### Dosya Kapatma

**Close** metodu ile dosyalar kapatılır.

Bu metodun en önemli parametresi kapatırken kaydedip kaydetmemeye yarayan **Savechanges** parametresidir.

Aşağıdaki kod ile, aktif dosyayı kaydederek kapatıyoruz.

Sub kapama()

Activeworkbook.Close savechanges: =True

End Sub

Open metodunda olduğu gibi, eğer bir Auto\_close makrosu varsa, bu makro Workbook.Close metodu ile tetiklenmez, dosya X işaretine basılarak kapatıldığında tetiklenir. Bunun için Open metodunda olduğu gibi şu kod eklenmelidir, ancak bu sefer close işleminden önce eklenmelidir.

ActiveWorkbook.RunAutoMacros xlAutoClose

ActiveWorkbook.Close Savechanges: =False

Aşağıdaki örnekte, açık olan tüm dosyaları kapatan ama kapatmadan önce kaydedip kaydedilmeyeceklerini soran bir kod bulunuyor. Tabi Personal.xlsb gibi gizli dosyalarda bu işlemi atlıyoruz, yoksa tüm Excel kapanır. Ayrıca henüz kaydedilmemiş dosyalar için ayrı bir işlem uyguluyoruz.

Sub tümdosyaları\_kapat()

Dim wb As Workbook

cevap = MsgBox("Kapatırken save edeyim mi", vbYesNoCancel)

If cevap = vbYes Then

A = True

ElseIf cevap = vbNo Then

A = False

Else

Exit Sub

End If

For Each wb In Application.Workbooks

Workbooks(wb.Name).Activate

If Windows(wb.Name).Visible = True Then

If InStr(ActiveWorkbook.Name, ".") = 0 Then 'henüz kaydedilmemiş bir dosyaysa

wb.SaveAs Filename: =Application.DefaultFilePath & "\" & wb.Name & ".xlsx", \_

FileFormat: =Application.DefaultSaveFormat, CreateBackup: =False

ActiveWindow.Close

Else

wb.Close savechanges: =A

End If

End If

Next wb

End Sub

#### Yeni Dosya Yaratma

Yeni bir dosya yaratacağımız zaman **Add** metodunu kullanırız. Bağımsız bir satırda kullanacağımız gibi aynı anda bir değişkene de atayabiliriz.

Sub yenidosya()

Dim wb as Workbook

Set wb=Workbooks.Add

End Sub

#### Dosya Kaydetme

Basit kaydetme işlemi **Save** metodu ile, farklı kaydetme işlemi ise **SaveAs** metodu ile yapılır. Save metodu parametre almaz, dosyayı sadece kaydeder. SaveAs metodunun birkaç parametresi vardır.

Tüm syntax şöyle: **SaveAs(FileName,  FileFormat,  Password, WriteResPassword, ReadOnlyRecommended,  CreateBackup, AccessMode, ConflictResolution, AddToMru,  TextCodepage, TextVisualLayout, Local)**

FileName ile dosyaya ne isim vereceğimizi belirtiriz, bu isim klasör ismini de içerebilir, klasör belirtilmezse Default kaydetme klasörüne kaydedilir.

FileFormat ile dosya formatının ne olacağına karar veririz.

Belli başlı dosya formatları şöyle olup, tüm listeye MSDN sayfalarından ulaşabilirsiniz. Parantez içindeki değerler enumaration sabitleridir, kodlarınızda parantez içi de dışı da kullanılabilir.

|  |  |
| --- | --- |
| İsim(Değer) | Açıklama ve uzantı |
| xlExcel12(50) | 2007 sonrasında binary format, xlsb |
| xlWorkbookDefault(51) | 2007 sonrasında gelen klasik format, xlsx |
| xlOpenXMLWorkbookMacroEnabled(52) | 2007 sonrasında gelen makrolu format, xlsm |
| xlExcel8(56) | 2007 sonrasında eski klasik format, xls |
| xlWorkbookNormal(-4143) | 2007 öncesinde klasik format, xls |

Dosya kaydetme işlemlerinde Excel versiyon kontrolünü de yapmak, zorunlu olmasa da, hataları ele alması açısından akıllıca bir yol olacaktır.

Aşağıda örnek bir kod bulunmaktadır.

Sub versiyonkontrol()

Dim wb As Workbook

Dim uzantı As String

Dim frmt As Integer

Set wb = ActiveWorkbook

If Val(Application.Version) < 12 Then

'2007 öncesini kullanıyorsunuzdur

uzantı = ".xls": frmt = -4143

Else

'2007 sonrasını kullanıyorsunuzdur

cevap = MsgBox("Eski formatta mı kaydetmek istiyorsunuz?", vbYesNo)

If cevap = vbYes Then

uzantı = ".xls": frmt = 56 'belki bu dosyayı 2003 kullanan bir alıcıya göndereceksinizdir

Else

uzantı = ".xlsx": frmt = 51

End If

End If

wb.SaveAs Filename: ="Yenidosya" & uzantı, FileFormat: =frmt

End Sub

Bir kaydetme metodu daha vardır, bu metod dosyanın o anda bulunan halini (kendi üzerinde kayıt işlemi yapmadan) başka bir yere kaydetmeye yarayan **SaveCopyAs** metodudur. Mevcut dosya üzerinde kayıt işlemi yapmadan, ama bu halini de saklaması adına güzel bir metoddur. Yani özetle, SaveAs açık dosyayı o andaki değişiklikleriyle kaydederken, SaveCopyAs, açık dosyayı aynen bırakır, buna herhangi bir kaydetme işlemi uygulamaz, ancak son değişiklikleri içeren halini farklı bir isimde kaydeder. Bu da SaveCopyAs'i **yedekleme amacına** hizmet etmesi için harika bir metod yapar.

Mesela aşağıdaki kod Ribbonunuzda kısayol tuşu olarak bile atayabileceğiniz güzel bir koddur.(FileDialog işlemlerini ayrıca göreceğiz)

Sub anlık\_yedekal()

'kodun sağlıklı çalışması için References içinde Microsoft Scripting Runtime kütüphanesi eklenmiş olmalı

'eğer bu kütüphane ekli değilse ve şuan eklemeyi tam bilmiyorsanız aşağıdaki satırı diğer iki set cümlesinin altına ekleyin

'Set fso = CreateObject("Scripting.FileSystemObject")

Dim adres As String

Dim isim As String

Dim fd As FileDialog

Dim wb As Workbook

Dim fso As New FileSystemObject

Set wb = ActiveWorkbook

Set fd = Application.FileDialog(msoFileDialogFolderPicker)

fd.ButtonName = "Seçin"

fd.Title = "Kayıt yeri seçin"

fd.InitialFileName = wb.path

If fd.Show = -1 Then

adres = fd.SelectedItems(1)

isim = InputBox("Dosya ismi ne olsun", Default: =fso.GetBaseName(wb.Name) & ".... yapmadan önceki yedek")

wb.SaveCopyAs (adres + "\" + isim + "." + fso.GetExtensionName(wb.Name))

End If

End Sub

##### Network üzerinde kaydetme işlemi

Network üzerinde bir yere kayıt yaparken bazen bir bug(hata) nedeniyle kaydetme işlemi ekranda takılı kalmaktadır. Bunun ne zaman olacağı asla kestirilememekte(%3-5 ihtimal diyebilirim) ve Microsoft da maalesef bu soruna bir çözüm bulabilmiş değildir. Eğer schedule edilmiş bir iş sırasında başınıza bu gelirse sonrasında çalışması gereken tüm makrolar durmaktadır, bu da gününüzün iyi geçmemesi için yeterli bir gerekçe olabilir.

Bununla beraber bu konuda bir çözüm bulunmaktadır. Sırayla şu adımlar uygulanmalıdır.

* Dosya önce yerel makinaya kaydedilir
* Sonra bu dosya kapatılır
* Sonra dosya kapalıyken yerel diskten networkteki klasöre kopyalanır
* Yerel diskteki dosya silinir

Bunun kod örneği aşağıdaki gibidir.

Sub çağıran()

'.....

'.....

hedef=ActiveWorkbook.FullName

saveas\_islemi(hedef)

End Sub

Sub saveas\_islemi(ByVal hdf As String)

Application.DisplayAlerts = False

ActiveWorkbook.SaveAs Filename: ="C: \geçici\geçici.xlsm", FileFormat: =xlOpenXMLWorkbookMacroEnabled 'burda dosya isminin ne olduğu önemli dğeil bile

Kaynak = ActiveWorkbook.FullName

ActiveWorkbook.Close 'filecopy yapmadan önce dosyayı kapatmak lazım yoksa izin vermiyor

FileCopy Kaynak, hdf

Kill Kaynak

Application.DisplayAlerts = True

End Sub

NOT: Eğer bu işlemin Workbook\_Open prosedürü içinde yapılması gerekirse ikinci bir geçici dosya daha yaratmak gerekecektir.

### İleri Seviye işlemler

#### Workbook kaydedilmiş mi kontrolü

Dosya üzerinde, son açıldığından beri bir değişiklik yapılıp yapılmadığını görmek için **Saved** özelliği kullanılır. True dönerse, son açılışından beri bir değişiklik yapılmadığını gösterir.

Bu property, özellikle Excel'i kapatırken işe yarar. Şöyle ki, diyelim ki belli bir saatte Excel'i kapatmak istiyorsunuz, kapatırken de o anda tüm açık dosyaları sanki kayıtlıymış gibi göstermeniz gereken durumlar olabilir. Olayı daha somutlaştırmak adına yine kendi işimden örnek vermek istiyorum.

Normalde işimi yaptığımı PC'm dışında **Application.Ontime** konusunda bahsettiğim bir de laptop'ım var, orda da scheduled makrolar çalışıyor. Bunların dışında bir üçüncü bilgisayar daha var ki, bu da bir televizyona bağlı. Televizyonda 10 dk'da bir refresh olan birkaç rapor var, oradan tüm departman anlık olarak bu rapor sonuçlarını görüyor.

Bu bilgisayarı her akşam 21: 00'de kapanacak, her sabah 08: 30'da da açılacak şekilde ayarladım. Kapama işlemini makroyla, sabah açma işlemini Windows Task Scheduler'a yaptırıyorum. Sabah açılır açılmaz, 21: 00'de kapatacak makroyu schedule ediyorum. Schedule etme işinin detaylarını **Application.Ontime** konusunda detaylıca göreceğiz.

Bu projeyi Veritabanı uygulamalarında **Hayalet Protokol(çok havalı isim dimi)** projesinde detaylıca ele alıyoruz. Şimdi sadece bizi ilgilendiren kısmının kodlarını inceleyelim.

Ön bir bilgi daha: Task Scheduler Excel’i açar açmaz, XLSTART klasöründeki schedule.xlsb dosyası da açılıyor. Bunun Workbook\_Open eventinde da birkaç dosyayı açan bir kod var. Açılan ilk dosya "Kredi raporu", aşağıdaki kodlar da bu kredi raporuna aittir.

NOT: Aşağıdaki prosedürlerde olay kaydı tutan **Logger** isimli fonksiyonu detaylı olarak **Dosya Okuma ve Yazma** **İşlemleri** konusunda göreceğiz. Siz sadece **Saved** özelliğinin kullanıldığı yere odaklanın.

'Öncelikle Workbook\_Open makrosuna bakalım

Private Sub Workbook\_Open()

'Öncül kodlar(tanımlamalar vs)

If Environ$("computername") = "A12345" Then 'TV bilgisayarıysa diye bakıyor. Çünkü bu dosyayı tüm departmandakiler açabilir. Sadece TVye bağlı bilgisayarda otomatik refresh devreye girmeli

'kodlar

Application.OnTime Now + TimeValue("00: 10: 00"), Procedure: ="TVRefresh" ’10 dk sonra tekrar çalışsın

'kodlar

End If

Logger "Giris", 0, "Giris yapildi ve 21: 00e 'kapat' schedule ediliyor"

Application.OnTime TimeValue("21: 00: 00"), Procedure: ="kapat"

'Diğer kodlar

End Sub

'-------------------------------------------------------------------------

'şimdi de Workbook\_BeforeClose'a bakalım

Private Sub Workbook\_BeforeClose(Cancel As Boolean)

On Error GoTo hata 'kapatma tuşuna basıp tüm bunlar yapıldıktan sonra hiç iptal edilecek schedule işlemi kalmıyor ama en son "şu dosyaları save edeyim mi" diye bir dialog çıkarıyor ya, buna cancel dediğimde Excel’de kalmış oluyoruz, sonra ikinci bir kapat dediğinde iptal edecek bir schedule olmadığı için hata oluyor, o yüzden hata kontrolü koydum

If Hour(Now) < 21 Then

Application.OnTime TimeValue("21: 00: 00"), Procedure: ="kapat", Schedule: =False '21den önce kapatılırsa kapat'ın schedulını iptal ediyoruz

Logger "Kapanis", 0, "21den once BeforeClose'a girildi ve SCH iptal edildi"

Else

Logger "Kapanis", 0, "21de BeforeClose'a girildi ve kapanıyor"

End If

Exit Sub

hata:

If Err.Number = 1004 Then Resume Next

End Sub

'---------------------------------------------------------------------------

'son olarak da kapat makrosuna bakalım, bizim esas ilgilendiğimiz kısım burası

Sub kapat()

On Error GoTo hata

If Environ$("computername") = "A12345" Then

'.....

For Each wb In Workbooks

wb.Saved = True 'işte Saved özelliğini burada kullanıyorum,

Next wb

Logger "OtoKapanma", 0, "kapat makrosu ile TV'de kapanıyor"

Application.Quit 'Burası ile de Excel kapatılıyor

Else

Logger "OtoKapanma", 0, "kapat makrosu ile kapanıyor"

Windows("Anlık Kredi Raporu.xlsm").Close savechanges: =False

End If

Exit Sub

hata:

Logger "Hata", Err.Number, "Kapat makrosunda hata: " & Err.Description

End Sub

Neden böyle yapıyoruz, bunu açıklayayım.

Kapat makrosu içine direkt Application.Quit dersem, Excel bana "şu şu dosyaları kaydedeyim mi" diye sorar ve ekran öylece kalır. Bu istediğim birşey değil.

Neden Workbook.Close deyip **Savechanges** özelliğine False atamıyorum? Çünkü böyle dersem kodun çalıştığı kredi dosyası kapanırsa(ki ilk o kapanacaktır çünkü ilk açılan dosya o) kalan dosyalar için kapatma işlemi uygulanmaz.

Tüm dosyaları kaydedip neden kapatmıyorum? Çünkü dosyalar readonly açılıyor, kaydedilemez; gerçi readonly açılmasa bile kaydetme işlemini TV bilgisayarında yaptırmazdım.

O yüzden dosyaların **Saved** özelliğine True atayıp, onları sanki kaydedilmiş gibi gösteriyorum ve böylece Excel'in bana soru sormamasını sağlıyorum.

Bu örnek, şuan için karışık gelmiş olabilir ancak genel mantığını anlamışsınızdır diye umuyorum. Anlamadıysanız bile sorun yok, Hayalet Protokol örneğini incelerken bu kısma tekrar gelip bakarsınız. O zaman daha iyi anlamış olacaksınız.

#### Açık dosyalar arasında dolaşma

Tüm açık dosyalarda dolaşmak ve onlarda işlem yapmak için aşağıdaki basit kodu kullanabilirsiniz.

Sub wbisimleri()

Dim i As Integer

For i = 1 To Workbooks.Count

'buraya diğer kodlar yazılır

Next i

End Sub

Bazen Personal.xlsb gibi dosyaları hariç tutarak işlem yapmak isteriz.  Örneğin, açık olan tüm dosyalarınızı kapatmak isterseniz yukardaki kodu alıp araya Workbooks(i).Close dersek, bu kodu yerleştirdiğimiz Personal.xlsb de kapandıktan sonra hiç açık dosya kalmayacağı için Excel de kapanmış olur. O yüzden gizli dosyalar hariç bakalım demeliyiz. Ancak Workbook nesnesinin **Visible** diye bir özelliği yok, bunun yerine Window nesnesinin bu özelliğini kullanacağız. Bu durumda kodumuz aşağıdaki gibi olur.

Dim wb As Workbook

For Each wb In Workbooks

If Windows(wb.Name).Visible = True Then

'diğer kodlar buraya

End If

Next wb

Açık dosyalar arasında Window nesnesnin metodları aracılığı ile de dolaşabiliriz. Window nesnesi birçok açıdan Workbook'a benzese de, bir workbook içnde birden fazla window olduğu durumlarda biraz farklılaşma yaşanabilir. Pencereler arasında dolaşma **ActiveWindow.ActivateNext** ve **ActiveWindow.ActivatePrevious** metodları aracılığı ile olur, ilki bir sonraki pencereyi(workbooku) aktive ederken ikincisi bir öncekini aktive eder. İki dosya arasında git gel yaptığınız durumlarda oldukça kullanışlıdır. Ancak birden fazla dosyanın açık olduğu durumlarda karışıklığa neden olacağı için kullanmanızı tavsiye etmem.

#### ActiveWorkbook & ThisWorkbook & Me

Çoğu zaman birbiri yerine kullanılabilecek olan bu terimler arasında küçük farklar bulunmaktadır. **ActiveWorkbook**, o anda aktif olan dosya iken **ThisWorkbook**, kodun çalıştığı dosyadır. Mesela, bizim ana kod dosyamız olan Personal.xlsb dosyası gizli bir dosya olduğu için genelde activeworkbook o olmayacaktır, ancak eğer ondaki bir kod çalışıyorsa ThisWorkbook o olacaktır.

Sayfa gibi bir nesnenin önünde Workbook ifadesi belirtilmediyse Excel bunun ActiveWorkbookun bir sayfası olduğunu düşünür. Bunun bir istisnası vardır: Eğer bir Workbook modülüne(ThisWorkbook) girildiyse ve burada bir Workbook adı belirtilmeden bir nesne mesela bir sayfa adı kullanılıyorsa, bu durumda bu nesne Activeworkbooka ait değil kodun çalıştığı workbooka ait olarak algılanır.

Başka bir örnek ise şöyle olabilir: Diyelim ki Kredi Format.xlsm diye bir dosyanız var, gece belli bir saatte çalışacak şekilde programlanmış olsun. Bu dosya ilgili saatte açıldığında, Workbook\_Open makrosu devreye girsin ve bazı kodları çalıştırsın. Kodların bir bölümünde PersonelBilgileri.xlsx diye bir dosyayı açıp bundan sicil kodlarının yanına personelin ismini getirsin ve eğer kredi dosyasında personelbilgi dosyasında olmayan bir sicil varsa bunu bu dosyaya eklesin. Bu durumda kredi dosyasından copy yapılıp personel dosyasına paste işlemi yapılacağı için son açık görünen dosya yani activeworkbook, personelsicil dosyası olacaktır. O yüzden paste işleminden hemen sonra Activeworkbook.Close denirse, personelsicil dosyası kapatılmaya çalışılır. Thisworkbook.Close denirse, aktif dosya personel dosyası olduğu halde kredi dosyası kapatılmaya çalışılır, zaten o kapanırsa kod da durur, ve personel dosyası açık bir şekilde bekler.

Bu örneği biraz da aşama aşama görelim, yazım tekrarı olmaması adına ThisWorkbook'un kod boyunca hep Kredi dosyası olacağını söyleyelim ve sadece Activeworkbook'un değişimini izleyelim:

* Kredi dosyası açılır açılmaz Workbook\_Open devreye girdi --> Activeworkbook=Kredi
* Personel dosyası açıldı-->Activeworkbook=Personel
* Personelden krediye lookup yapılacak, kredi aktive edilir--> Activeworkbook=Kredi
* #N/A gelenler Personel dosyasına eklenecek, personel aktive edildi ve yeni siciller yapıştırıldı --> Activeworkbook=Personel.
* Önce personel dosyasını kapatmak için Activeworkbook.Close Savechanges: =True denir
* Akabinde ThisWorkbook.Close Savechanges: =True denir ve kredi dosyası da kapanır

Bir de **Me** ifadesi vardır. Eğer kodumuz ThisWorkbook modülü içindeyse bu ifade, ThisWorkbook anlamında kullanılabilir. (Ancak Me'nin başka anlama geleceği durumlar da olabilir. Eğer, kodumuz bir sayfa modülü içindeyse mesela Sheet1 modülündeyse, Me=Sheet1'dir. Onun dışında Me'nin en çok kullanıldığı yer sanırım UserFormlardır. Bu durumda da Me, formun kendisi olmaktadır. Bunu Userformlar bölümünde göreceğiz.)

#### Dosya ismi ve adresi/klasörü

Dosyanın ismine **Name** özelliği ile ulaşırız. İsimden kastımız uzantı dahil isimdir. **Ör: "Krediler.xlsx".** Worksheet'in Name özelliğinden farklı olarak bu özellik Readonly'dir(Nesne model tanımında "Sets" yok, sadece "Returns" vardır), yani dosya adı dosya açıkken değiştirilemez. Ancak **SaveAs** veya **SaveCopyAs** ile dosyaları farklı isimde kaydetme imkanımız olabilir.

Name özelliği dosyanın sadece adı ve uzantısını verirken, **Fullname** özelliği tüm path'i de verir. **Ör: "C: \Users\Volkan\Desktop\deneme.xlsx"**

Bazen de dosya adını uzantısı olmadan almak isteriz, mesela uzantısız ismi alıp bu ismin sonuna bir ek ekleyip sonra tekrar uzantıyı eklemek isteyebilirsiniz. Bunun için birkaç yöntem var. Hepsini de göstermeye çalışacağım, başka yöntemler de bulunabilir tabiki, VBA'de bir sonucu elde etmenin binbir türlü şekli var sonuçta.

**1.yöntem: FileSystemObject** objesinin **GetBaseName** özelliğini kullanmak. Bu kanımca en ideal yöntemdir, zira her versiyon ve uzantıda işe yarar. Bunu yukarda anlık\_yedek al makrosunda kullanmıştık.

CreateObject("Scripting.FileSystemObject").GetBaseName(ActiveWorkbook.name)

**2.yöntem:** Uzantının ne olduğunu biliyorsak, bunu "" yani sıfır uzunluklu metinle replace ederiz.

isim = Replace(ActiveWorkbook.Name, ".xlsx", "")

**3.yöntem:** Dosya isminde nokta işareti olmadığından eminsek, noktanın yerini bulup oraya kadar olan ismi almaktır.(Dosya isminde nokta varsa bu yöntem işe yaramaz)

isim=Left(ThisWorkbook.Name, (InStrRev(ThisWorkbook.Name, ".", -1, vbTextCompare) - 1))

**4.yöntem:** Uzantı kadar karakteri hariç tutup soldan yeterli sayıda karakter seçmek. Bu örnekte bir de versiyon kontrolü yapılabilir.

Dim i As Integer

If Application.Version = 12 Then 'Excel 2007 ve sonrasıysa

If ActiveWorkbook.FileFormat=56 then 'xls uzantılıysa

i=4

Else

i=5

End If

Else

i = 4

End If

uzantısızisim= Left(ActiveWorkbook.Name, Len(ActiveWorkbook.Name) - i)

Ve isterseniz bütün bunları bir fonksiyona da atayabilirsiniz.

Function uzantısızisim() As String

Dim i As Integer

If Application.Version >= 12 Then

If ActiveWorkbook.FileFormat=56 then

i=4

Else

i=5

End If

Else

i = 4

End If

uzantısızisim= Left(ActiveWorkbook.Name, Len(ActiveWorkbook.Name) - i)

End Function

Dosyanın bulunduğu klasör ismine **Path** özelliği, uzantısına da yine isminde olduğu gibi birkaç yolla ulaşılabilir, ben örnek olarak sadece **FileSystemObject** objesinin **GetExtensionName** özelliğini söyleyeyim, diğer yolları siz düşünün.

FileSystemObject nesnesine ayrıca bakacağımız için burada daha fazla detaya girmiyorum.

#### Workbook Protection

Excel'in Review menüsünden konulup kaldırılan şifre işlemleri elbette ki VBA ile de yapılabilmektedir.

Mesela genel kullanıma açık olan bir dosyanızda otomatik refresh işlemleri olduğunu düşünelim. Dosya gece çalışıp içindeki sorgular refresh edilecek ve bazı copy paste işlemleri olacak diyelim. Dosya genel kullanıma açık olduğu ve kimsenin gizli sayfalardaki veriye ulaşmasını istemediğiniz için dosyayı korumaya almış olabilirsiniz. Ancak gece dosya otomatik açıldığında işlemlerin öncesinde korumayı kaldırmalı, işler bitince tekrar korumayı aktive etmelisiniz. İşte kodumuz:

Private Sub Workbook\_Open()

ThisWorkbook.UnProtect (1234)

'işlemler yapılır

ThisWorkbook.Protect (1234)

ThisWorkbook.Close Savechanges: =True

End Sub

Bir dosyada koruma olup olmadığını görmek için de **ProtectStructure** özelliği kullanılır. Aşağıdaki örnekte belirli bir grup dosyada toplu connection şifre değişikliği gerçekleşecek ancak öncesinde her dosya için protection kontrolü yapılıyor.

'Belirli dosyaları files isminde bir collectiona atadım ve bu collection içinde dolanarak çeşitli işlemler yapacağım

For Each file In files

Workbooks.Open Filename: =file

Set wb = ActiveWorkbook

If wb.ProtectStructure = True Then

protectlimi = True

wb.Unprotect (1234)

End If

For Each cn In wb.Connections

If InStr(cn.ODBCConnection.Connection, eski) > 0 Then

cn.ODBCConnection.Connection = Replace(cn.ODBCConnection.Connection, eski, yeni)

End If

Next cn

'protection olanlarda tekrar koyalım

If protectlimi = True Then wb.Protect (1234)

protectlimi = False 'tekrar false yapıyorum ki, döngüye tekrar girdiğinde True olarak gitmesin

Next file

#### Diğer

Workbook'un başka özellik ve metodları da bulunmaktadır. Ben burada önemli olduğunu düşündüklerimi vermeye çalıştım. Sizler diğer üyeleri araştırabilir ve kendiniz deneyebilirsiniz.

Bununla beraber RefreshAll ve Connection gibi üyeleri ise VeriTabanı işlemlerinde ayrıca ve detaylıca ele alacağım için burada değinmedim.

## Nesnelerin Efendisi - Application

### Giriş

Önceki bölümlerden hatırlayacağınız üzere, Excel, nesneler hiyerarşisi üzerine kurulmuş bir **Nesne Modeline** sahiptir ve işte şimdi göreceğimiz **Application** nesnesi de hiyerarşinin en tepesinde bulunur.

**Application** nesnesi Excel’in ta kendisidir ve bu yüzden de default nesnedir. Bu şu demek, bazı durumlarda bu ifadeyi yazmanıza gerek olmadan buna ait özellik ve metodları kullanabiliriz. Yani **Application.ActiveWorkbook** yazmak ile **ActiveWorkbook** yazmak arasında hiçbir fark yoktur. Ancak bazı durumlarda da Application'ı açıkça yazmak gerekir.

Eğer Excel’in kendisiyle ilgili bir işlem olacaksa o zaman Application'ı açıkça yazmamız gerekir. Mesela Excel’den çıkış için **Quit** metodunu kullanmak, Excel’in ekrandaki boyutlarını ayarlamak gibi. Bununla beraber benim tavsiyem, Application'ı her durumda yazmanızdır, ancak olur da internette araştırma yaparken Application'ın yazılmadığını görürseniz de şaşırmayın. Aşağıda özellik ve metotların tanıtımında Application'ın yazılması gereken durumlar için bunu açıkça yazdım, diğer durumlar için yazmadım, ancak kod örneklerinde size verdiğim tavsiyeyi tuttum ve Applicationu hep yazmaya çalıştım.

Bu arada herhangi bir nesnenin Application özelliğini kullanarak da bu Application nesnesini elde edebiliriz. Daha teknik bir ifadeyle, bazı nesnelerde bulunan Application özelliği(propertysi) Application tipinde değer döndürür, yani Application nesnesi elde edersiniz. Tabi bazen kullandığımız nesne Excel Nesne Modeline ait bir nesne değil de mesela bir Outlook nesnesi olabilir, böyle bir durumda Application propertysi kullandığınızda dönen değer tabiki Excel değil, Outlook olacaktır. Bunları Diğer Ofis uygulamalarıyla çalışmak bölümünde göreceğiz.

### Genel Görünüm ve Uygulama Seviyesi İşlemleri

Excel'in genelini ilgilendiren birçok üye mevcuttur. Bunların birçoğu **File>Options**'tan ulaşabileceğiniz ayarların VBA karşılıklarıdır. Önemlilerine, daha doğrusu kısa ve orta vadede kullanma ihtimaliniz olan üyelere bir bakalım.

**Application.DisplayFullScreen özelliği**: Boolean tipinde değer döndürür. True atandığında Ribbon olsun durum çubuğu olsun hiçbir şey göstermez, sadece hücreler ve formül çubuğu görünür.

**Application.DisplayFormulaBar özelliği:** Bu da Boolean tiplidir. False atanırsa formül çubuğu gösterilmez. Bunu bazen formülleri göstermemek için(protection yaparak da sağlanır) bazen de Displayfullscreen özelliği ile birlikte, ekranda maksimum alanda yer açmak kullanılır.

**Application.DisplayStatusBar özelliği:** Bu da Boolean tiplidir, En alttaki durum çubuğunu gösterir veya gizler. (Application.StatusBar özelliği ile karıştırılmamalıdır, bu ikincisinde durum çubuğunda yazan metni alırız veya metin yazarız.)

**Application.DisplayScrollbars özelliği**: Bu da Boolean tiplidir, scrollbarları gösterir veya gizler.

**Application.Interactive özelliği**: Boolean döndürür. Diyelim ki çok fazla copy-paste yapan bir makronuz var, kodunuz da uzun sürüyor, beklerken o sırada Word veya Outlook'ta vakit geçireyim dediniz. Outlookta yazdığınız bir metni kesip başka bir yere kopyalamaya karar verdiniz, ancak tam az önce de Excel VBA kodunuz da bir kesme işlemi yapmıştı, siz şimdi clipboarda Outlook metnini almış oldunuz ve kodunuz hızlıca akıp geçti, ve paste işlemini yaparken Excel’den aldığı parçayı değil, Outlooktaki metni yapıştırdı. İşte böyle bir durum olmasın diye bu tür işlemlerinizin olduğu kodlarınızın başına bu özelliği yazıp False değerini atayabilir, kodun sonunda bunu yine True'ya döndürebilirsiniz.

Application.Interactive = False

'kodlar

Application.Interactive = True

**Application.Quit metodu:** Excel’den çıkış için kullanılır.

Bu arada Excel’den çıkış yapılmasını yakalayacak bir event yok maalesef. Bunu farklı yöntemlerle tespit eden bazı makaleler gördüm ama oldukça karışık olduğu için buraya almak istemedim. Ben şahsen, bunun yerine Personal.xlsb dosyasının kapanıp kapanmadığını bu dosya içindeki Workbook\_Beforeclose olayı ile yakalıyorum, bunun kapanması demek zaten birçok durumda Excel’in kapanması demek oluyor, ki bu da işimi görüyor. Bunların detayını Olaylar(Events) bölümünde göreceğiz.

**Application.StatusBar**: Görev çubuğuna mesaj yazmak için kullanılır. Özellikle kullanıcıları hem bilgilendirmek hem de bilinçli bir şekilde mesaj kutusu çıkarmak istemediğinizde faydalıdır. (Bazen schedule edilmiş işlerde arkadan gelen kodların takılmasını engellemek için, bazen de kullanıcılardan gelen "Bu kadar mesajbox çok can sıkıcı" itirazlarını ele almak için). Ancak unutulmamalıdır ki, MsgBox kadar da dikkat çekici değildir, hatta bazı durumlarda verdiğiniz mesaj gözden kaçabilir de. O yüzden kritik mesajları MsgBox ile vermenizi tavsiye ederim. Aşağıda bu özelliği ProgressBar olarak nasıl kullanıyoruz, onu da göreceğiz.

#### Son Söz

**Object Browser** veya MSDN üzerinden Uygulama seviyesinde yapılabilecek daha bir çok ayarlama olduğunu görebilirsiniz, buraya önemli olduğunu düşündüklerimi aldım, diğerlerini siz de araştırabilirsiniz. Bunların çoğu Excel Options üzerinden yapacağınız ayarlamalara denk gelir. Ör: Autorecover ayarı, autocorrect ayarı, dosyalar açıldığında linkleri update etsin mi ayarı gibi.

### Kod hızlandırıcılar

Aşağıdaki üç özellik kodlarınızın başında False sonunda True olarak ayarlandığında performans kazanımı sağlar.

**Application.ScreenUpdating özelliği:** Uzunca bir makro çalışırken ekranın bir gidip geldiğini, titrediğini görmüşsünüzdür(veya göreceksinizdir), hele hele farklı workbooklar arasında gidip gelme söz konusu ise bu durum çok daha göze çarpar. Aslında tüm bu ekran hareketleri, genel süreci uzatan bir rol oynar, zira işlemciniz o sırada ekranı güncellemekle de ilgilenmektedir. O yüzden bu ekran hareketini kapatarak kodunuzu hızlandırabilirsiniz. Bunu da bu özelliğe False değeri atayarak yapıyoruz. Kod bitmeden hemen önce açmayı unutmayın tabi.

Kod çok uzun sürüyorsa ScreenUpdating=False durumunda kullanıcılar Excel’in kitlendiğini düşünebilir, o yüzden arada bir hareket göstermek iyi olabilir. Bunu da Doevents metodu ile yapabiliriz.

'Genellikle bir döngü içinde mantıklıdır

Application.ScreenUpdating=False

Do Until oldumu = True

'ara kodlar

DoEvents 'burda ekran tazelenir

Loop

**Application.DisplayAlerts özelliği:** Kodunuz çalışırken Excel bize bazı uyarılar çıkarabilir, bunlar da genelde can sıkıntısı yaratabilir. Özellikle schedule edilmiş makrolarınız varsa ve bunların birinde bir sayfa silme, veya var olan dosya üzerine yazma gibi size uyarı çıkaran kodlar varsa, bu özelliğe False atamazsanız ekran ilk uyarıda takılı kalır ve sizin bir cevap vermenizi bekler. Eğer bir seri schedule edilmiş kodunuz varsa, böyle bir durum kabul edilemez. O yüzden en faydalı bulduğum özelliklerden biri budur. Bu özelliğe false atandığında uyarılara varsayılan cevap verilir ve kod devam eder. **Kod bittiğinde de bu özelliğe otomatik True değeri atanır.**

Ancak bu özellik iki durumda işe yaramaz.

* Mesaj kutularında. Özellikle schedule programınız varsa içinde mesaj kutusu kullanmamaya çalışın. Hata yönetimi işlemlerinde bile kullanmayın, onun yerine kendinize mail gönderebilir, StatusBarı kullanabilir veya Log kaydı tutabilirsiniz.

İçinde başka dosyalara link olan dosyalar açıldığında Linkleri update edeyim mi sorusu. Burda DisplayAlerts yerine **Workbook.Open** metodunun **UpdateLinks** parametresi kullanılır. Gerçi bunu da aşmanın bir yolu var, ama dikkatli kullanılmasında fayda var.**Application.AskToUpdateLinks** özelliğine False atanırsa linkler otomatik güncellenir, ve bu soru karşımıza çıkmaz, böylece her Workbook.Open metodunda tek tek Updatelinks özelliğine değer girmek zorunda kalmayız. Ama bazı durumlarda otomatik update olmasın isterseniz bunu kullanmak yerine Workbook.open metodunun **Updatelinks** parametresini kullanın.

**Application.EnableEvents özelliği:** DisplayAlerts ile birlikte en çok değer verdiğim bir diğer özellik de budur. Hatta QuickAccess barda bu özelliği True ise False, False ise True yapan bir düğmem bile var. Bunu  biraz sonra açıklayacağım. Öncelikle ne işe yarar ona bakalım.

Bu özellik, herhangi bir event(olay) tetiklenmesin diye kullanılır. Event’leri ayrıca göreceğiz. Ancak örnek olarak bir dosyanın açılmasını veya sayfada bir değişiklik yapılmasını gösterebiliriz. Bunlar programatik açıdan anlamlı olaylardır ve gerçekleştiklerinde bazı kodların devreye girmesini isteyebiliriz.

Bu bağlamda, bu özelliğin İki tür kullanım şekli olabilir.

* Bir programın en başına false, en sonuna tekrar true olacak şekilde. Böylece tüm kod boyunca hiçbir olay tetiklenmez.
* Bir döngü içinde satır silme, hücre değeri değiştirme gibi bir işlem vardır, ve sayfa modüllerin birinde Worksheet\_change eventiniz de vardır, sadece bu event tetiklenmesin diye ilgili döngünün başına ve sonuna konur. Böylece döngüden çıkıldığında diğer eventlerin tetiklenmesine imkan verilmiş olur.

Ben çok faydalı bulduğum bu 3 özelliği bir prosedüre bağladım, ve birçok makroya girerken bunlara(bazen sadece ikisine) false değerini atıyorum, koddan çıkarken de tekrar true değerine döndürüyorum. Fonksiyon ve kullanım şekil aşağıdaki gibidir:

'Ana prosedürü

Public Sub AlertUpdatingEvent(a As Boolean, u As Boolean, Optional e As Boolean = True)

With Application

.DisplayAlerts = a

.ScreenUpdating = u

.EnableEvents = e

End With

End Sub

'Prosedürü çağırma şeklim

AlertUpdatingEvent False, False 'Son parametreyi eklemediğimi için default değeri olan True atanır. Böylece sadece uyarıların çıkmasını ve ekran titreşimini engellemiş ama eventler çalışsın istemiş oldum

### Hesaplama, Zamanlama(Scheduling) ve Bekle(t)me

Bu başlıktaki konular her zaman olmamakla birlikte genelde bir arada kullanılmaktadırlar, en azından bir kullanım yakınlığı vardır diyebiliriz.

#### Hesaplama işleri

**Application.Calculation özelliği:** Excel’in formüller için hesaplama yöntemini seçmenizi sağlar. Excel’in bu özelliğini bildiğinizi varsayıyorum, bilmiyorsanız öncesinde mutlaka konuyu internette araştırın.

Bu özelliğin alabileceği 3 enumaration değeri var.   
  
**xlCalculationAutomatic** : Varsayılan değer budur. Herhangi bir hücrede değişiklik olduğunda tüm workbooklarda formüller yeniden hesaplanır.   
**xlCalculationSemiautomatic** : Table'lar dışında herşey otomatik hesaplanır.   
**xlCalculationManual** : Hesaplama işlemi kapalıdır. Kullanıcı hesaplama yapana kadar da öyle kalır.  
  
Özellikle büyük formüllü dosyalarda bir makro çalıştıracaksanız ve herhangi olumsuz bir etkisi olmayacaksa öncesinde hesaplama kapatılıp makro bitmeden hemen önce de tekrar açılabilir.

Application.Calculation = xlCalculationManual

'burada diğer işler yapılır

Application.Calculation = xlCalculationAutomatic

**Calculate metodu**: Tüm workbooklardaki yeni, değişmiş ve volatil(araştırın) formüllerin hesaplanmasını sağlar(o anda Manuel hesaplama seçimi yapıldıysa anlamlıdır, aksi halde zaten formüller hesaplanmıştır ve gerek yoktur). Calculation property'sinin aksine bunda Application denmesine gerek yoktur. Örnek biraz aşağıda bulunmaktadır.

**Application.CalculationState özelliği**: Hesaplamanın ne durumda olduğunu gösterir. Alabileceği değerleri bir kodla görelim.

Sub CalcState()

If Application.CalculationState = xlDone Then 'enumeration değeri 0

MsgBox "Hesaplama Bitti"

ElseIf Application.CalculationState = xlPending Then 'Görev çubuğunda "Calculate" yazar 'enumeration değeri 2

MsgBox "Tetiklendi ama henüz hesaplama başlamadı"

Else 'xlCalculating 'enumeration değeri 1

MsgBox "Hesaplama devam ediyor 'Görev çubuğunda %sel bir oran görünür

End If

End Sub

**Done** ve **Calculating** gayet aşikar fakat **Pendingi** tam olarak anlamamış olabilirsiniz. Hani bazen Excel manuel hesaplama modundayken, formüllerden birine baz teşkil eden bir hücreyi değiştirdiğinizde en alttaki durum çubuğunda **Calculate** yazar, bazen de dosyanızda çok sayıda formül varsa Excel bu kadar formülle başa çıkamaz ve en altta yine Calculate yazar. İşte bu durum xlPending durumudur. Böyle durumlarda Excel’de hesaplama yapmak için Formulas menüsünden Calculate demek veya F9'a basmak gerekir.   
**NOT**: Bir de kısırdöngülü formüllerde Calculate yazdığını görürsünüz, bu da bir xlPending durumudur ancak onun çözümü aşağıdakiler değil, kısırdöngüye neden olan formülü düzeltmektir.

İnternette birçok forumda yaygın bir kullanım örneği olarak aşağıdaki kod parçası verilir. Deniyor ki, "kodunuz çalışmaya başlamıştır, büyük bir hesaplama yapıyordur, ancak daha hesaplama bitmeden bir sonraki satıra geçer, bu da hatalı sonuçlar neden olabilir, o yüzden aşağıdaki kod ile kodunuzun aşağı satıra geçmesini engellersiniz".

Application.Calculate 'hesaplamaya başladınız

Do While Application.CalculationState <> xlDone

DoEvents

Loop

'kodun kalan kısmı

Halbuki Calculate metodu asynchoronus değildir, yani hesaplama bitmeden zaten bir sonraki satıra geçmez. O yüzden yukarıdaki tavsiye bence anlamsızdır. Ancak bir şekilde(forumlarda yardım isteyen diğer kişilerin başına gelen çok özel durumlarda, artık neyse o özel durumlar bilemiyorum) böyle birşey olduğunu fark ederseniz bu kodu kullanabilirsiniz.

Bu arada şu farkı iyi anlamanız gerekiyor; Calculate metodunu uyguladığınızda sanki Excel’de F9'a basmış veya Calculation menüsünden Calculate butonuna basmış gibi olursunuz ve formüller yeniden hesaplanır ancak sayfanız o an hala Manuel modda kalmaya devam eder ve sonraki aşağı/sağa formül kaydırma işlemleri sonucunda formüller hesaplanmaz. Halbuki Calculation özelliğine xlAutomatic atayarak hem hesaplamayı açmış olurusunuz hem de statüyü kalıcı olarak otomatiğe çevirmiş olursunuz ve sonraki formül kaydırmalarda formüller hemen hesaplanır. Hangisi ihtiyacınıza uygunsa onu kullanmalısınız. Eğer ki geçici bir hesaplama yapmak istiyorsanız Calculate metodunu, kalıcı hesaplama için ise Calculation özelliğini kullanabilirsiniz.

Önemli bir husus da şudur; Application, Worksheet ve Range nesneleri için var olan Calculate metodu Workbook için bulunmamaktadır. Ancak aşağıda gibi bir kod ile sadece Activeworkbook'un Calculation işlemini yapabilirsiniz.

Sub CalcBook()

Dim ws As Worksheet

Application.Calculation = xlManual

For Each ws In ActiveWorkbook.Worksheets

ws.Calculate

Next

Set ws = Nothing

End Sub

**Application.CalculateFull**: Otomatik veya Manuel modda olun fark etmez, tüm formüllü hücreleri yeniden hesaplar. Calculate metodundan farklı olarak, sadece yeni, değişmiş ve volatil formülleri değil, tüm formül içeren hücreleri tekrar hesaplar. Bu yüzden genelde(her zaman değil) normal Calculate metoduna göre daha yavaştır. Durum çubuğunda ısrarla Calculate yazıyorsa yani xlPending durumundan bir türlü çıkamıyorsanız bunu kullanabilrsiniz. Klaveye kısayolu **Ctrl+Alt+F9**'dur.

**Application.CalculateFullRebuild:** Bu metod CalculateFull ile aynı işi yapıyor gibi görünüyor, Excel 2007 ve sonrası kullanıcıların çok kullanacağı bir metod değildir. Özetle şunu diyebilirim ki, 2007 öncesi versiyonlarda aşırı formülden dolayı hesaplama zinciri bozulduysa ve F9 yaptığınız halde Excel hesaplama yapmıyorsa bu metod işe yarayacaktır. Ancak sanki tüm hücrelere formülleri tekrar girmek gibi iş yaptığı için CalculateFull'e göre biraz daha yavaştır.

**Application.CalculationInterruptKey**: Hesaplamanın hangi tuşla iptal edileceğini söyler. Bunun pratik kullanımı, Personal.xlsb'nin Workbook\_Open makrosu içine yazma şeklindedir. Ben şahsen sadece ESC tuşuna(xlEscKey) basıldığında hesaplamanın iptal edilmesini istiyorum, size de bunu öneririm. Zira eliniz yanlışlıkla bir ok tuşuna değse bile o anda %90larda olan calculation tekrar %0dan başlayacaktır.

##### Genel öneriler

Önce normal Calculation yapın. Sonra state kontrol edin, hala xlPendingse CalculateFull uygulayın.

Kod hızlandırıcılar bölümündeki 3 özelliğe bazen bu Calculation'ı da ekleyerek daha hızlı kod çalıştırabilirsiniz. Ancak kullanımı konusunda dikkatli olmak gerekir, zira arada bir yerlerde formül çekme/uzatma ve sonra Copy-Paste işlemi varsa Calculation sonucunda hatalı durumlar oluşabilir.

Sub Calculationlar()

AlertUpdatingEvent False, False, False

Application.Calculation = xlCalculationManual

'kodlar buraya gelir

'arada bir açmak gerekebilir

ActiveSheet.Calculate 'duruma göre Application.Calculate veya Range("...").Calculate

If Application.CalculationState=xlPending then Application.CalculateFull

'tekrar kapatalım

Application.Calculation = xlCalculationManual

'diğer kodlar

'çıkışta tekrar eski haline getiriyoruz

AlertUpdatingEvent True, True, True

Application.Calculation = xlCalculationAutomatic

End Sub

Son olarak Calculate işlemlerinin VBA ve Excel ilişkilerini tekrar şöyle bir özetleyelim:

|  |  |  |
| --- | --- | --- |
| İşlem | Excel | VBA |
| Tüm workbookları hesaplatmak | Calculation>Calculate(veya F9) | Application.Calculate |
| Aktif sayfayı hesaplatmak | Calculation>Calculate Sheet(Shift+F9) | ActiveSheet.Calculate |
| Aktiveworkbook hesaplatmak | - | Döngü içinde Sheet.Calculate |
| Belli bir range'i hesaplatmak | - | Range.Calculate |
| Full hesaplama yapmak | Ctrl+Alt+F9 | Application.CalculateFull |

##### Calculation için örnek bir senaryo

Şimdi diyelim ki departmanınızdaki kişilerin kullanması için çok sayfalı ve çok formüllü bir Excel dosya hazırladınız. İlk sayfada tek sayfalık bir karne/skorkart tarzı birşey var, diğer sayfalarda ise toplu listeler. Hepsi de datayı gizli bir sayfadan alıyor.

Liste sayfalarında çok fazla satır ve sütun ve hep SUMIFS tarzı formüller olduğu için bunlarda sort veya filter işlemleri çok ağır olmaktadır, zira bu iki işlem de calculation tetikleyicisidir. Çözüm şöyle olabilir:

Dosyanın Workbook\_Activate(Neden Workbook\_Open olmadığını az sonra belirteceğim) eventine dosya açılır açılmaz Calculation'ı xlManual yapan kodu ekledim, ve bir mesaj kutusu ile bunu kullanıcıya bildiriyorum.(MsgBox sinir bozucu gelirse statusbara da yazdırabilirsiniz). Karne sayfasına gelince ise Worksheet\_Change eventine yazdığım kod ile sadece belli hücreler değiştiğinde hesaplama yapmasını sağlıyorum. Kullanıcı olur da o sırada başka dosyalarda işlem yapmak isterse Workbook\_Deactivate eventine calculationı tekrar otomatik yapan bir kod yazdım ki, kullanıcı o sırada Calculationın kapatıldığını unutup diğer dosyalarda formül uzatma gibi işler yaparsa hep aynı sonucun yazdığını görüp şaşırmasın. Hatta ortalama bir kullanıcı Excel’in Calculation özelliğinden bihaberdir bile diyebiliriz. Kodlar şöyle:

Private Sub Workbook\_Activate()

Application.Calculation = xlCalculationManual

Application.StatusBar = "Dosya aktive olduğu için Calculation yine geçici olarak Manuel yapıldı"

End Sub

Private Sub Workbook\_Deactivate()

Application.Calculation = xlCalculationAutomatic

Application.StatusBar = "Başka dosyayı açtığınız için Calculation tekrar otomatik yapıldı"

End Sub

**Application.Volatile metodu**: Bu metoda UDF konusunda değineceğimiz için burada ayrıca bahsetmiyorum.

#### Erteleme ve bekleme

**Application.OnTime metodu**: Kitap boyunca zaman zaman programlanmış işlerden veya İngilizce tabiri ile işleri schedule etmekten bahsediyorum, mutlaka dikkatinizi çekmiştir. İşte bu işi bu harika metod ile başarıyorum. Genel kullanım şeklini aşağıda veriyorum ama bu konuyla ilgili uzunca bir örneğe veritabanı programlama bölümünde **Hayalet Protokol** örneğinde ele aldım.

Şimdi metodun genel syntaxına bakmadan önce görevini açıkça belirtelim: Bir prosedürün belirli bir anda çalışmasını sağlar, ki bu tanım bize onu asıl amacı dışında(ama faydamıza olacak şekilde) kullanacağımızı da söylemektedir, yani **Wait** ve **Sleep** metodları yerine. Bunlara da hemen bu metottan sonra değineceğiz.

**Syntax**: **ApplicationObject.OnTime(EarliestTime, ProcedureName, LatestTime, Schedule**

Tam açıklaması şöyle oluyor. ProcedureName ismindeki makro EarliestTimeda başlasın, o sırada başka bir makro çalışıyorsa veya Excel’i meşgul eden başka birşey varsa da LatestTime'a kadar çalıştırmayı denesin. Eğer LatestTime belirtilmezse, Excel’in meşguliyeti bitene kadar bekler ve sonra çalıştırır. Yani eğer, "Kod, programladığım saatten en geç 1 saat içinde çalışsın, yoksa çalışmasının bir anlamı yok, çünkü o rapor artık işe yaramaz olur" dediğiniz bir durum varsa bu parametreyi "EarliestTime + 1 saat" olarak belirtebilirsiniz, aksi durumda boş bırakın. Schedule parametresi default değeri True'dur ve genelde yazılmaz, schedule ettiğiniz bir prosedürü iptal etmek için bu değere **False** atarsınız.

**Önemli Not**: Excel’den her çıkış yaptığınızda, tüm schedule programı sonlanır. Eğer, recursive(tekrarlı) yani bittikten sonra yeniden schedule edilen programınız varsa Excel’i hep açık bırakmanız gerekir, ki benim bilgisayarımda olan budur. Ve bence bu metottan verim almanın en güzel yolu onu recursive bir şekilde kullanmaktır. Şimdi küçük bir örnek bakalım, siz sonra Veritabanı Programlama konusunda ele alacağımız daha detaylı örneği incelersiniz.

Sub ontimeornek()

Application.OnTime Now + TimeSerial(0, 0, 3), "mesajver" 'Asynhronous metoddur

MsgBox "beklemeden çalıştım"

End Sub

Sub mesajver()

MsgBox "selam"

End Sub

Örnekte gördüğünüz üzere mesajver makrosunun çalıştırılacağı zamanı Run tuşuna bastıktan 3 sn sonra çalıştıracak şekilde parametrik verdim. Yani burada spesifik bir saat belirtmek yerine, şimdiye(Now) referansla bir saat de verebiliyoruz. Örnek gösterimler şöyle olabilir.

Application.OnTime "22: 30: 00"

Application.OnTime Now + TimeValue("00: 10: 00") 'Şimdiden 10 dk sonra

Application.OnTime Now + TimeSerial(0,10,0) 'Bu da aynı. TimeSerial'de saat, dakika ve saniye virgülle ayrılır

**Application.Wait metodu**: Programın belirli bir süre durmasını(beklemesini) sağlar. Peki neden? Neden programınızın bir süre durmasını bekleyesiniz ki? İşte örnek senaryolar olmayınca maalesef makro öğrenimi çok zor olmaktadır. Önce gelin nasıl kullanılacağına, sonra nedenine bakalım.

Sub bekle()

Application.Wait (Now + TimeValue("0: 00: 10")) 'Synhronous metoddur

MsgBox "bekleyip çalıştım"

Call mesajver

End Sub

Sub mesajver()

MsgBox "selam"

End Sub

Bu metod Boolean döndürdüğü için belirli bir zaman geçip geçmediğini kontrol etmek için de kullanılır.

If Application.Wait(Now + TimeValue("0: 00: 10")) Then '10 sn geçtiyse. =True demeye gerek görmeyebiliyoruz, önceki konuları hatırlayacak olursanız

Application.Speech.Speak "Zaman doldu" 'Evet, Excel 2013ten itibaren artık konuşuyor

End If

Dikkat: Bu metod kullanılırken çok dikkat etmek gerekir, zira ilgili süre geçene kadar Excel kitlenir.

Şimdi de örnek bir senaryo düşünelim. Diyelim ki bir makronuzu sabah/gece 5'e schedule ettiniz: Kodunuz bir veritabanını güncel veri gelmiş mi diye 10 dk'da bir tarıyor, ve sonunda 5:40’ta yani 4. seferinde güncel datayı gördü ve hemen çekti, bikaç işlem yaptı, 5:43’te işi bitti ama kod devam ediyor, ettiği yerde başka bir veritabanı bağlantısı yapacak, ama siz biliyorsunuz ki o veritabanı 6:30’da doluyor, 5:43’te buraya bağlanmaya çalışırsa güncel olmayan veriyi alabilir, işte böyle bir durumda kodu 6:30’a kadar bekletmek gerekebilir.

Application.Wait "06:30:00"

Hatta bunu bir de saat 6:30’dan önce  mi diye kontrol etmek lazım, eğer ilk veritabanını sorgulanması 5:40 değil de 6:30’dan sonraya kaldıysa ikinci veritabanı için beklemelik bir durum olmayacaktır, o yüzden Wait kullan**mama**k gerekir, aksi halde ertesi sabah 6:30'a kadar Excel’iniz bloke olur.

Bir diğer örnek durum da şu olabilir. Veritabanı işlemlerinde göreceğiz gerçi ama **RefreshAll** gibi **asyncrohnous**(sonraki satıra geçmek için beklemeyen) bir metod çalıştığında kod okuma devam eder. Eğer tüm refresh işleminin bitmesini beklesin istiyor ve tahminen refreshin ne kadar süreceğini biliyorsanız ilgili süre kadar bekletebilirsiniz.

'Önceki kodlar

Me.RefreshAll

Application.Wait Now + "00: 30: 00" '30 dakika yeterlidir

'diğer kodlar

**Sleep Metodu:** Bu metod VBA metodu olmayıp Windows fonksiyonudur, bu yüzden bunu kodumuzun başına import etmek gerekir. Wait ile aynı işi yapar, tek farkı, milisaniye cinsinden paremetre almasıdır. İmport işlemi dahil tam bir kod örneği aşağıda bulunmaktadır. (import işlemleri hakkında detaylı bilgi <https://msdn.microsoft.com/en-us/library/office/gg278581.aspx> sayfasında mevcuttur)

'bu kısım import kısmıdır, kod sayfasının en tepesine yazılır

#If VBA7 Then

Public Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As LongPtr) '64 Bit Sistemler için

#Else

Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long) '32 Bit Sistemler için

#End If

'bu kısım esas kod kısmı

Sub sleepmetodu()

MsgBox "Başlıyorum"

Sleep 10000 '10bin mili saniye yani 10 sn bekliyoruz

MsgBox "Süre bitti"

End Sub

Yukarda belirttiğimiz gibi, **OnTime** metodunu da amacı dışında bir nevi bekleme amacı olarak kullanabiliriz, ancak Wait ve Sleepte kod akışı o satırda duruken OnTime'da durmaz devam eder, o yüzden OnTime eğer bu amaçla kullanılacaksa ilgili prosedürün son satırı olmasında fayda var.

Sub tetikleyici()

'çeşitli kodlar

Application.OnTime Now + TimeSerial(1,0,0), "makrom"

End Sub

'şimdi bu durumda 1 saat sonra makrom makrosu başlayacak. Ekran 1 saat boyunca serbest.

Sub makrom()

'çeşitli kodlar

End Sub

Bunların dışında bir de farklı bir yaklaşım var ki, bunda da OnTime'da olduğu gibi bekleme süresince Excel’e erişimimiz açık durmaktadır, zira bunda bir döngü içinde **DoEvents** metodu kullanılmaktadır.

Dim newTime As Date

newTime = Now + TimeValue("00: 00: 10")

Do While Not Now >= newTime

DoEvents

Loop

MsgBox "selam

**DoEvents:** MSDN açıklaması çok yüzeysel maalesef, ben yine de bu açıklamayı vereceğim, sonra nerelerde kullanıldığını söyler ve birkaç örnek gösterirsem daha anlaşılır olur.

**MSDN açıklaması:** Program akışını işletim sistemine verir.

Şimdi MSDN'nin açıklamasını biraz daha açalım. Bir Windows uygulamasında aynı anda onlarca program çalışabilmektedir. Sizin Excel VBA kodunuz işlemciyi çok fazla bloke ederse Windows buna sinirlenebilir, hatta bakar ki Excel’den ses seda yok, bunun kitlendiğini düşünüp kapatmaya çalışabilir, çünkü belleksiz kalan diğer gariban programlar çalışsın diye düşünür. İşte böyle durumlarda araya bir **DoEvents** sokmak gerekebilir, ki Windows Excel'in yaşadığını düşünsün. Peki genel olarak nerelerde kullanırız ona bakalım:

* Büyük döngülerde araya girebilmek için(özellikle kitlenme durumları yaşıyorsanız)
* ScreenUpdating=False olduğunda kullanıcı Excel’in kilitlendiğini sanmasın diye ekran tazelemede
* Bir şartın olmasını beklerken

Do Until beklenen=true

DoEvents

Loop

Bu yukardaki ilk iki maddeyi içeren bir örnek verelim. Uzun bir döngüsel işleminiz var diyelim, hızlı çalışsın diye ScreenUpdating=False yaptınız. Kullanıcı program kitlendi sanmasın diye, **DoEvents** metodu Statusbarı bir nevi **Progressbar** gibi kullanmamızı sağlayacak. Tabi bu aşağıdaki örnekte %1den %100e giden bir progressbar yapmış olduk ancak siz isterseniz başka ölçüler kullanabilirsiniz. Mesela 20 bölgeli bir bankada her bölgenin işi 1 dk sürüyorsa, her döngü sonuna "20 bölgede " & i & " adedinde işlem tamam" gibi bir metin yazdırabilirsiniz.

Sub doeventprogressbar()

Dim i As Long

Dim bas As Double

bas = Timer

Application.ScreenUpdating = False

For i = 1 To 100000 '100.000in 100lük dilimlere böldüğümüzde her bir bölümün büyüklüğü

Cells(1, 1) = i

If i Mod 1000 = 0 Then

DoEvents

Application.StatusBar = "%" & i \* 100 / 100000

End If

Next i

Application.ScreenUpdating = True

MsgBox Round(Timer - bas) & " sn sürdü"

End Sub

Bu arada bu koddan screenupdatingi çıkarın, programın ne kadar yavaşladığını göreceksiniz.

### Dosya ve Klasör işlemleri

Bazen kullanıcıdan, üzerinde işlem yapılacak bir dosya veya klasör seçmesini isteriz. Bazı durumlarda seçilen dosya ile sadece işlem yapılırken bazen dosyanın açılması sağlanır.

Bu kısımda anlatılan konular genel olarak, seçilen dosyayı açma veya bir şekilde dosya/klasör ismi elde etme amacıyla kullanılan işlemlerle ilgilidir. Daha genel olarak tüm dosya işlemlerini Dosya İşlemleri bölümünde ele alıyor olacağız.

Kullanıcıdan dosya/klasör bilgisi istemenin en ilkel yolu bunu bir inputboxla sormak olacaktır, ancak şükür ki VBA'de bunu yapmamızı sağlayan daha iyi yöntemler var. Şimdi bunlara bakalım:

**Application.FileDialog özelliği**: 2002 yılında gelen bu özellik bundan daha önce var olan **GetOpenFilename** ve **GetSaveAsFileName** özelliklerinin gelişmiş halidir. O yüzden bu ikisinin artık çok kullanmaya gerek yok, ama başka kodlarda görmeniz durumunda ne olduğunu bilmeniz için onlara da kısaca değineceğiz. Bu özelliğin FileDialogType şeklinde tek bir parametresi vardır ve o da **MsoFileDialogType** sabitlerinden biri olabilir. Bunlar:

**msoFileDialogFilePicker:** Dosya seçtirir, path dahil tam ismini döndürür(Ör: "C: \Hedefler\Satış\2016.xlsx"

**msoFileDialogFolderPicker:** Klasör seçtirir, path dahil tam ismini döndürür(Ör: "C: \Hedefler\Satış"

**msoFileDialogOpen:** Açılacak dosyayı seçtirir(onu açmaz, sadece seçtirir)

**msoFileDialogSaveAs:** Farklı Kaydet dialog kutusunu açar, dosyayı kaydederken ismin ne olacağını girmenizi sağlar(dosyayı kaydetmez, sadece isim ve adres belirlersiniz)

Bu property ile **FileDialog** nesnesi elde edilir. Bu nesnenin de kendi metod ve özellikleri vardır. Genel olarak önce bu tipte bir değişken yaratmak ve ona atama yapmak intellisense açısından uygun olacaktır.

Bu nesnenin iki metodu var, pratikte en çok kullanacağınız metodu **Show** metodudur. Bir seçim yapıldıysa -1(True) döndürür, seçim yapılmadan işlem iptal edilirse 0(False) döndürür.

Seçim sonucu True ise **Execute** metodunu yazarak duruma göre uygun işlemi de yaptırabilirsiniz. Ancak bunun yerine Workbook.Open veya Workbook.SaveAs gibi metodlar da kullanılabilir. Execute yazmak basit görünse de arka planda seçim tipi ve sonucunu karşılaştıran bir koşullu yapı barındırdığı için performans sorununa neden olabilir, özellikle büyük kodlarda. O yüzden doğrudan Workbook metotlarını kullanmanızı tavsiye ederim.

Önemli propertyler ise şöyledir.

**AllowMultiSelect**: Çoklu seçim yaptırma imkanı verir, sadece dosyalar için geçerlidir, klasörlerde çoklu seçim yapılamaz.

**Title**: Dialog kutusunun başlığını değiştirebilirsiniz.

**InitialFileName**: Seçim yaptırırken sık kullanılan bir dosya/klasör varsa default olarak bunu seçtirebilirsiniz, SaveAs yaparken de yine aynı mantıkla default bir adres ve isim belirleyebilirsiniz.

**Filters**: Hangi tür dosyaların gösterileceğini belirlersiniz.

**SelectedItems**: Seçilen dosyaların/klasörlerin tam adresini verir. Tek seçim yapıldıysa SelectedItems(1) şeklinde kullanılır.

Şimdi ilk olarak dosya seçme örnekleriyle başlayalım, sonra da diğerlerine geçelim.

Aşağıdaki örnekte dosya seçtiriyorum ve seçilen dosyaları siliyorum. Bu arada **With .. End With** yapısını nasıl kullandığıma dikkat edin.

Sub Dosyaislemlerim()

Dim i As Byte

Dim fd As FileDialog

Set fd = Application.FileDialog(msoFileDialogFilePicker)

With fd

.AllowMultiSelect = True

.Title = "Silinecek dosyaları seçin"

.Show

For i = 1 To .SelectedItems.Count

Kill .SelectedItems(i) 'buradaki kill metodunu sonra göreceğiz, şimdilik sadece dosya silmeye yaradığını bilin

Next

End With

End Sub

Dialog kutusunu seçim yapmadan kapadığımızda bunu anlayan bir kontrol noktası koyalım. Bu kontrol noktasını doğrudan metodun sonuç değeri ile yapıyoruz.

Sub Dosyaislemlerim()

Dim i As Byte

Dim fd As FileDialog

Set fd = Application.FileDialog(msoFileDialogFilePicker)

With fd

.AllowMultiSelect = True

.Title = "Silinecek dosyaları seçin"

If .Show = 0 Then

MsgBox "Seçim yapmadan çıkış yaptınız"

Exit Sub

End If

'iptal edilirse buraya gelmeden program sonlanır, çünkü Exit Sub denildi

For i = 1 To .SelectedItems.Count

Kill .SelectedItems(i)

Next

End With

End Sub

Şimdi de diğer 3 tip için de örnekler yapalım. Önce dosya aç:

Sub Dosyaislemlerim()

Dim fd As FileDialog

Set fd = Application.FileDialog(msoFileDialogOpen)

With fd

'.AllowMultiSelect = True

.Title = "Açılacak dosyayı seçin"

.InitialFileName = "C: \deneme.xlsx"

If .Show = True Then

.Execute 'veya Workbooks.Open (.SelectedItems(1))

End If

End With

End Sub

Şimdi default dosya adı belirtmeyelim, kullanıcı seçsin ama çok kalabalık dosya türünü barındıran bir görüntü de olmasın, sadece Excel dosyaları olsun.

Sub Dosyaislemlerim()

Dim fd As FileDialog

Set fd = Application.FileDialog(msoFileDialogOpen)

With fd

'.AllowMultiSelect = True

.Title = "Açılacack dosyayı seçin"

'.InitialFileName = "C: \inetpub\wwwroot\ee\yuklemeler\pivot - data.xlsx"

.Filters.Clear 'varsayılan olakra 20 tane dosya tipi var, bunları temizlzeyelim

.Filters.Add "Excel dosyaları", "\*.xls\*"

.Filters.Add "Tüm dosyalar", "\*.\*"

If .Show = True Then

.Execute 'veya Workbooks.Open (.SelectedItems(1))

End If

End With

End Sub

Eğer fark ettiyseniz msoFileDialogFilePicker ve msoFileDialogOpen ifadelerinin her ikisini de dosya açmada kullanabiliriz. Hatta dosya silme işlemi için bile msoFileDialogOpen kullanılabilir. Yukarıdaki msoFileDialogFilePicker ile yapılmış dosya silme örneğinde kendiniz deneyip görebilirsiniz. O halde neden iki ayrı ifadeye gerek var diye düşünüyor olabilirsiniz. Buna maalesef benim de cevabım yok. O kadar araştırdım ancak bir açıklama bulamadım. Bulduğum zaman bu paragrafı güncellerim.

Şimdi de folder işlemlerine bakalım

Sub folderislemleri()

Dim fd As FileDialog

Set fd = Application.FileDialog(msoFileDialogFolderPicker)

With fd

.AllowMultiSelect = True 'buna True dense bile çoklu seçim yaptırmaz

.Title = "klasör sçein"

If .Show = True Then

MsgBox .SelectedItems(1) & " klasörünü seçtiniz"

End If

End With

End Sub

Son olarak da SaveAs işlemi yapalım.

Sub Dosyaislemlerim()

Dim fd As FileDialog

Set fd = Application.FileDialog(msoFileDialogSaveAs)

With fd

.InitialFileName = "C: \deneme.xlsx"

If .Show = True Then

.Execute 'veya Workbooks.Saveas ile

End If

End With

End Sub

**Application.GetOpenFilename ve Application.GetSaveAsFilename**: Yukarıda bahsettiğim gibi yeni kodlarınızda bu iki metod yerine FileDialog yöntemini kullanmanızı tavsiye ederim. Bunları sadece geriye dönük uyumluluk adına ve karşınıza bunları içeren bir kod geldiğinde anlamanız için bilmeniz gerektiğini düşünüyorum.

**Syntax: GetOpenFilename( GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)**

Bunda FileDialogda olduğu gibi bir nesne yaratmaya gerek yok, doğrudan kullanılabilir. Bundan dönen değer üç şey olabilir. Seçim yapılmadıysa False(Boolean) veya tek bir dosya seçim yapıldıysa dosyanın tam adını(path dahil) veren bir string ya da çoklu seçim yapıldıysa bir dizi. O yüzden dönüş tipi varianttır ve tanımlanırken de böyle tanımlanmalıdır.

Yine FileDialogda olduğu gibi burda da doğrudan dosya açma veya kaydetme yok sadece dosya ismi elde edilir, sonrasında ayrı bir satırda dosya açma işlemi yapılır.

Sub getopenfilenameornek()

Dim filtreler As String

Dim başlık As String

Dim dosya As Variant

filtreler = "Excel dosysaları(\*.xls\*),\*.xls\*, Tüm dosylar (\*.\*),\*.\*"

başlık = "Açılacak dosyayı seçin"

dosya = Application.GetOpenFilename(filtreler, 5, başlık, , True)

If IsArray(dosya) Then 'multi parametresini true belirlediğimiz içn öncelikle dizi olup olmadığını kontrol etmemiz lazım

For Each a In dosya

Workbooks.Open (a)

Next a

Else

If dosya = False Then

MsgBox "seçim yapılmadı"

Else

Workbooks.Open dosya

End If

End If

End Sub

**SaveAsFileName** de bunun aynısı olup sadece save etmeye yarıyor.

**FindFile**: Bu da,  GetopenFileName ve FileDialog gibi yine Open File dialog kutusunu getirir ancak geriye bir dosya adı döndürmez. Eğer seçim yapıldıysa dosyayı açar ve True değerini döndürür, seçim olmadıysa False döndürür. Bunu diğerleriyle bir anlam bütünlüğü var diye buraya aldım ancak açıkçası pratikte nasıl bir kullanımı olur bilmiyorum, şahsen ben şimdiye kadar hiç kullanmadım.

### Diğer üyeler

**Application.ActivateMicrosoftApp metodu:** Bu metod, başka bir MS Office uygulamasını açar. Eğer halihazırda ilgili uygulama açıksa onu aktive eder yoksa yenisini yaratır ve açar. Ancak bunu bu şekilde doğrudan kullanmak yerine ilgili Office uygulamasını obje olarak yaratıp onun Nesne modeline ulaşmak istemeniz durumunda ise(ki daha çok bu yöntemi kullanacaksınız) farklı bir yöntem kullanılır. Bunu da Diğer Uygulamalarla Çalışmak bölümünde görebilirsiniz. Bu metoda dönecek olursak aşağıdaki örnek kodda Word uygulaması açılmakta.

Application.ActivateMicrosoftApp xlMicrosoftWord

**Application.Inputbox metodu**: Bunu daha önce inceledik.

**Application.OnKey metodu**: Excel, bir makroyu kaydederken bize bunu bir kısayol tuşuna atayıp atamayacağımız konusunda bir imkan sunar, ancak bunun bir sınırı vardır ki o da sadece **Ctrl** tuşunu kullanmak zorunda olmamızdır ve bu da bir noktadan sonra yetersiz kalmaya başlıyor. İşte bu noktada OnKey metodu yardıma koşuyor, bununla istediğiniz tuş kombinasyonlarına atama yapabiliyorsunuz ve bu tuşlara bastığınızda bir olay tetiklenmiş gibi istediğiniz makro çalışmaya başlıyor.

Bu tuş kombinasyonu, sadece mevcut Excel oturumunda geçerli olmaktadır. Excel, kapatılıp tekrar açıldıktan sonra kullanılamazlar. Süreklilik kazandırmak için bunları Personal.xlsb dosyanızın Workbook\_Open makrosuna yazabilirsiniz.

Ör:

Private Sub Workbook\_Open()

Application.CalculationInterruptKey = xlEscKey

Application.OnKey "+^{F}", "Calculationlar"

End Sub

OnKey metodunu, Çeşitli Windows veya Office kısayol tuşlarını kontrol etmek için de kullanabilirsiniz. Mesela Cut/Copy işlemlerini engellemek için kullanabilirsiniz. Bunu da yine Workbook eventleri ile birlikte kullanmak gerekiyor. Bununla ilgili örnek biraz daha uzun olduğu için onu Workbook Eventleri sayfasına aldım.

**Application.Version:** Bu property ile Excel’in versiyonunu öğreniyoruz. Böylece kullanıcının Excel versiyonuna göre davranışımızı değiştirebiliyoruz. Mesela 2010 versiyonu ile birlikte gelen Slicer'larla ilgili bir işlemi 2007 ve öncesi kişilerde yapmaya çalışırsak hata alırız. Keza, Slicerlar 2010'da geldi ama sadece Özet tablolarda kullanılmak üzere gelmişti. Table'lar üzerinde kullanımı 2013 versiyonuyla geldi. Bu yüzden bir Table üzerinde Slicer kullanımı olacaksa yine hata alınır.

Şimdi Excel’in konuşması özelliğini kullanan başka bir örnek düşünelim. Bu özellik 2013 versiyonuyla geldiği için versiyon numarası 15.0'dır. Diyelim ki, ortak kullanım için bir Add-in yaptınız ve bu add-inde Kokpit adında tüm raporlara ulaşmayı sağlayan bir UserForm var. Kullanıcıların raporlara Kokpit üzerinden ulaşmasını istiyorsunuz, çünkü buradan ulaştıklarında raporlar Readonly açılıyor. Böylece siz raporlarda bir düzenleme yapmak istediğinizde kimsede açık bulunmamış oluyor ve düzenlemelerinizi rahatlıkla yapabiliyorsunuz. Ancak bazı yaramaz arkadaşlar dosyalara ortak alandan ulaşmaya çalışabilir. İşte onlar için ThisWorkbook modülünün Workbook\_Open makrosuna aşağıdaki kodu yazabilirsiniz.

Private Sub Workbook\_Open()

'diğer kodlar(Kullanıcının siz olması durumunda aşağıdaki kodun çalışmamasını sağlayacak kodlar dahil, şimdilik kafa karıştırmasın diye bunları atladım)

If Not Me.ReadOnly Then

If Val(Application.Version) >= 15 Then

Application.Speech.Speak ("Hey. Bana ortak alandan değil, Kokpit formu üzerinden gir") 'Konuşarak iletişim

Else

MsgBox "Hey. Bana ortak alandan değil, Kokpit formu üzerinden gir" 'MsgBox ile iletişim

End If

Logger "Bilgi", 0, "Ortak alandan girmeye calisma"

Me.Close savechanges: =False

End If

End Sub

## Neler Öğrendik

Excel’in 4 temel nesnesi üzerinde detaylıca çalışmayı öğrendik. Bu kısım en önemli kısımlardan biriydi. Bir sonraki bölümde tüm programlama dünyasının belkemiğini oluşturan algoritma akışlarına gireceğiz. Bunların bir kısmını ister istemez bu bölümde kullanmış olduk, sonraki bölümde ise daha detaylı inceleyeceğiz.