



Whitepaper Implementatie contextafhankelijke opdrachtbalken in MS Excel

Auteur: Emiel Nijhuis

Gepubliceerd: 15 april 2010

Inleiding

Binnen MS Office worden werkbalken, menubalken en snelmenu's via VBA bestuurd als één type object: opdrachtbalken. Een Office-toepassing kan gebruikmaken van eigen opdrachtbalken om functionaliteit te ontsluiten voor gebruikers. Dit geldt met name voor Excel-addin's die niet over een zichtbare interface beschikken in de vorm van een werkblad.

Uiteraard is het net als in andere Office-programma's mogelijk om zelf een nieuwe werkbalk te maken. Aan deze werkbalk voeg je vervolgens opdrachtknoppen toe waar je de macro's aan koppelt. In een professionele Excel-toepassing voldoet een dergelijke werkbalk echter niet:

- Het is niet mogelijk om via de user interface een item toe te voegen aan de menubalk.
- Via de user interface zijn de belangrijkste opdrachtknop-eigenschappen niet te benaderen. Denk hierbij aan de eigenschappen Parameter, Enabled of Tag.
- Hierdoor vertonen opdrachtknoppen geen gedrag: ze passen zich niet aan binnen een bepaalde context. Bijvoorbeeld: controls mogen in specifieke situaties niet beschikbaar zijn.

Vandaar dat creatie, manipulatie en opruiming van toepassingsopdrachtbalken via VBA zal moeten plaatsvinden. In Office-applicaties is dit echter een bewerkelijk proces. Een aantal redenen hiervoor zijn:

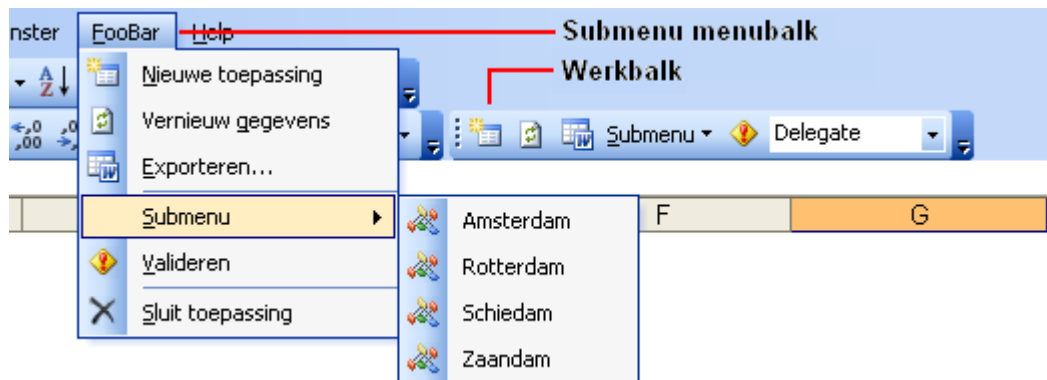
- Bij een addin van enige omvang zal er een menubalk-item en een werkbalk beschikbaar moeten zijn. De toepassingscontrols binnen de twee opdrachtbalken zijn veelal hetzelfde maar dienen wel per opdrachtbalk te worden toegevoegd.
- Een gebruiker kan op elk moment werkbalken en opdrachtknoppen verwijderen. Dit moet worden afgevangen binnen de code om foutmeldingen te voorkomen.
- Een addin kan op verschillende manieren worden geopend en gesloten; automatisch vanuit de opstartfolder, als invoegtoepassing via het gebruikersmenu of als autonoom bestand. Hier moet rekening mee worden gehouden bij creatie en opruiming van de opdrachtbalken.
- De positie en (on)zichtbaarheid van werkbalken kan door de gebruiker worden bepaald. Deze eigenschappen dienen te worden bewaard in geval van een addin-toepassing. Zodoende wordt voorkomen dat de gebruiker ze telkens opnieuw moet instellen bij opening van Excel.
- Het contextafhankelijk maken van opdrachtknoppen is verantwoordelijk voor de meeste code met betrekking tot de aansturing van de controls. Vooral de besturing van de beschikbaarheid van opdrachtknoppen levert de meeste hoofdbrekers op. Per opdrachtknop moet worden vastgesteld wanneer een gebruiker hem mag zien en er op mag klikken.

Probleemstelling

In dit artikel wordt de volgende situatie uitgewerkt: Een Excel-addin richt zich op een vanuit de addin toe te voegen werkmap. Door middel van een toepassingspecifieke menubalk-item en werkbalk kunnen bewerkingen op deze werkmap worden uitgevoerd. De beschikbaarheid en gedrag van de controls binnen deze opdrachtbalken dienen contextafhankelijk te zijn; Er bestaan drie contexten:

1. Context is Excel-applicatie; Hoe dienen de control zich te gedragen als er een event binnen Excel wordt afgevuurd?
2. Context is werkmap; de controls dienen zich automatisch aan te passen als gevolg van Excel-events indien de werkmap waar de addin zich op richt actief of inactief is.
3. Context is werkblad; hetzelfde geldt als een specifiek werkblad binnen deze werkmap (in)actief is.

Hoe implementeer je in een Excel-addin deze functionaliteit op een gestructureerde, flexibele, robuuste en gebruiks-/ onderhoudsvriendelijke wijze?



Oplossing

Het antwoord ligt in de toepassing van de combinatie van de volgende technieken:

1. Polymorfe implementatie van de opdrachtbalk-objecten
2. Opzetten van een logisch en hiërarchisch objectmodel
3. Het gebruik van een class om applicatie-events af te vangen en te implementeren
4. Zoveel mogelijk zoveel (context-afhankelijke) code centraliseren en hergebruiken

1. Polymorfe implementatie van de opdrachtbalk-objecten

Zoals al aangegeven willen we een toepassings specifiek submenu in de menubalk en een werkbalk toevoegen. Beide objecten fungeren als container voor dezelfde opdrachtknoppen maar zijn zelf van een ander type: een werkbalk is van het type `CommandBar` en een menubalk-item is van het type `CommandBarControl`. Deze verschillende objecten moeten echter dezelfde routines en properties implementeren om manipulaties op de `CommandBar(Controls)` uit te kunnen voeren. Vandaar dat beide objecten resideren in een eigen wrapper-class:

- `clsMenuBarControl`: Deze class implementeert de events, routines en eigenschappen van een toepassings specifiek submenu in de menubalk.
- `clsCommandBar`: De events, routines en eigenschappen met betrekking tot een toepassings specifieke werkbalk worden in deze class geïmplementeerd.

Beide classes implementeren de abstracte interface class `ICB`. `CB` staat dan ook zowel voor `CommandBar` (werkbalk) als voor `CommandButton` (menubalk-item).

Declaratie:

```
Private moMenuBarCtrl As ICB  
Private moCommandBar As ICB
```

Instantiatie:

```
Set moMenuBarCtrl = New clsMenuBarControl  
Set moCommandBar = New clsCommandBar
```

De class `ICB` bevat de volgende routines en eigenschappen:

- `Sub AddObject()`: Voegt het toepassings specifieke object toe.
- `Sub DeleteObject()`: Verwijderd het toepassings specifieke object.
- `Function ObjectExists()`: Checkt of het object bestaat. Retourneert een Boolean.
- `Sub InitializeControls()`: Initialiseert de `Enabled`-eigenschap van alle controls in het object.
- `Function FindControlByTag(sTag As String) As CommandBarControl`: Retourneert een `commandbarcontrol` binnen het opdrachtbalk-object op basis van de tag-waarde.
- `Sub AlterControl(sTag As String)`: Implementeert de wijziging van een specifieke control op basis van de control-tag.
- `Sub EnableControl(sTag As String, bEnabled As Boolean)`: Maakt een control (on)beschikbaar.
- `Sub ShowControl(sTag As String, bVisible As Boolean)`: Maakt een control (on)zichtbaar.
- `Property Get Object()`: Retourneert het eigenlijke object (`CommandBar` of `CommandBarControl`).



Een voorbeeld van polymorfisme m.b.t. de twee wrapper-classes is de implementatie van de functie FindControlByTag. Deze wordt in de classes verschillend ingevuld:

clsCommandBar:

```
Private Function ICB_FindControlByTag(sTag As String) As CommandBarControl
    Set ICB_FindControlByTag = ICB_Object.FindControl(Tag:=sTag)
End Function
```

clsMenuBarControl:

```
Public Function ICB_FindControlByTag(sTag As String) As CommandBarControl

    Dim cbc As CommandBarControl

    ' Type CommandBarControl beschikt niet over de routine 'FindControl'
    For Each cbc In ICB_Object.Controls
        If cbc.Tag = sTag Then
            Set ICB_FindControlByTag = cbc
            Exit For
        End If
    Next
End Function
```

Binnen de classes clsCommandBar en clsMenuBarControl wordt ook de wijze van toevoegen en verwijderen van van resp. de CommandBar en menubar-Control verschillend opgelost. Een menubalk staat doorgaans bovenaan de taakbalk. Een gebruiker zal dit niet willen wijzigen. Vandaar dat ons menubalk-item kan worden toegevoegd en verwijderd bij instantiatie resp. opruiming van het object moMenuBarCtrl:

```
Implements ICB

Private Sub Class_Initialize()
    ICB_AddObject
End Sub

Private Sub Class_Terminate()
    ICB_DeleteObject
End Sub
```

Bij het object moCommandBar ligt dat anders. De werkbalk is in eerste instantie 'floating' gegeneerd en getoond. Vervolgens kan het door de gebruiker ergens op de taakbalk zijn gedockt. Het is onwenselijk dat de gebruiker dit moet doen telkens wanneer hij Excel opent. Daarom wordt er bij instantiatie van moCommandBar eerst gekeken of de werkbalk al bestaat is. Indien dit zo is dan werd deze werkbalk bij het afsluiten van de vorige Excel-instantie niet verwijderd maar geïnitieerd. Zodoende kan een referentie naar deze werkbalk worden gezet:

```
Implements ICB

Private Sub Class_Initialize()

    If Not ICB_ObjectExists Then
        Call ICB_AddObject
    End If
End Sub

Private Sub Class_Terminate()

    If ICB_ObjectExists Then
        Call ICB_InitializeControls
    End If
End Sub
```

De routine ICB_ObjectExists wordt ook aangewend om te voorkomen dat het statement 'On Error Resume Next' gebruikt moet worden bij de bepaling of een object bestaat. Gebruik van dit statement wordt niet aanbevolen en maakt debugging lastig (in combinatie met andere Excel-toepassingen).



2. Opzetten van een logisch en hiërarchisch objectmodel

Afgezien van de reeds besproken classes `clsCommandBar`, `clsMenuBarControl` en `ICB` worden de volgende classes onderscheiden:

- `clsTool`: deze class implementeert de te ontsluiten addin-functionaliteit en fungeert als een container voor de classes `clsCommandBar` en `clsMenuBarControl`.
- `modCommon`: de module `modCommon` bevat globaal benaderbare constanten en objecten.
- `ThisWorkbook`: dit object handelt workbook-level events af en het bevat de routines aangeroepen vanuit de opdrachtbalken.
- `clsApp`: de class `clsApp` fungeert als een container voor application-level events en implementeert het contextafhankelijke gedrag van de `CommandBarControls`.

clsTool

Vanwege encapsulatie-overwegingen is het uiteraard raadzaam toepassingspecifieke functionaliteit in aparte classes onder te brengen. In deze toepassing is dat de class `clsTool`.

Bij instantiatie van het `goTool`-object worden de opdrachtbalk-objecten `moMenuBarCtrl` en `moCommandBar` geïnstantieerd. Wanneer `goTool` wordt opgeruimd geldt dat ook voor deze objecten.

Verder bevat `goTool` het object `moSheet`: een referencetype variabele naar het specifieke werkblad waar de addin zich op richt in de toe te voegen werkmap.

```
Private moCommandBar As ICB
Private moMenuBarCtrl As ICB
Private moSheet As Worksheet

Private Sub Class_Initialize()
    Set moMenuBarCtrl = New clsMenuBarControl
    Set moCommandBar = New clsCommandBar
End Sub

Private Sub Class_Terminate()
    Set moMenuBarCtrl = Nothing
    Set moCommandBar = Nothing
End Sub
```

modCommon

Het is goed gebruik om geen routines in een module te implementeren. Afgezien van de globaal benaderbare constanten bevinden zich hier de objecten `goTool` en `goApp`:

```
Public goTool As clsTool
Public goApp As clsApp
```

ThisWorkbook

Er worden drie `Workbook`-events geïmplementeerd in dit object. Twee daarvan richten zich op instantiatie en het opruimen van de twee globale objecten bij opening resp. sluiting van de addin.

```
Private Sub Workbook_Open()
    Set goApp = New clsApp
    Set goTool = New clsTool
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Set goApp = Nothing
    Set goTool = Nothing
End Sub
```

Het derde event `Workbook_AddinUninstall` verwijdert de werkbalk wanneer een gebruiker via de menu-optie `Extra;Invoegtoepassingen...`; de addin uit het geheugen heeft verwijderd. Merk op dat het event `Workbook_AddinInstall` niet is geïmplementeerd; alle noodzakelijke initialisaties worden namelijk al uitgevoerd in het event `Workbook_Open`.



Verder bevat het object de private routines die worden aangeroepen vanuit de opdrachtknoppen. Deze routines roepen de corresponderende functies aan in goTool, bijvoorbeeld:

```
Private Sub Validate()  
    goTool.Validate  
End Sub
```

clsApp

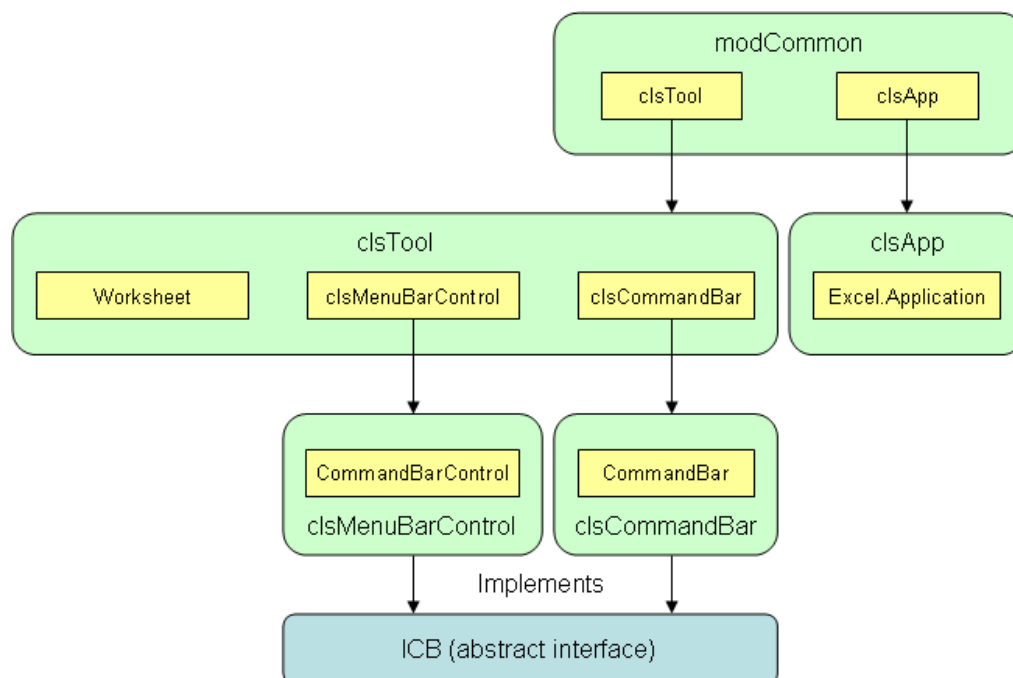
Om events buiten het eigen Workbook af te kunnen vangen wordt in class clsApp een Excelapplicatie-variabele gedeclareerd en geïnstantieerd met het keyword ' WithEvents':

```
Private WithEvents moApp As Excel.Application  
  
Private Sub Class_Initialize()  
    Set moApp = Excel.Application  
End Sub
```

Op deze wijze is de addin in staat op applicatieniveau events op te vangen, bijvoorbeeld:

```
Private Sub moApp_SheetActivate(ByVal Sh As Object)  
    [Een werkblad in één van de geopende werkmappen is geactiveerd]  
End Sub
```

De class-hiërarchie van de toepassing ziet er dus als volgt uit:



3. Het gebruik van een class om applicatie-events af te vangen en te implementeren.

De hierboven besproken class clsApp vangt op applicatieniveau events op. Vandaar ook dat in deze class veel van het contextafhankelijke gedrag van de opdrachtknoppen wordt geïmplementeerd. Per applicatie-event dienen echter wel alle gedefinieerde contexten te worden geëvalueerd; Welke manipulaties moet er op welke controls worden uitgevoerd binnen de context van werkblad, werkmap en applicatie? De volgende constructie wordt toegepast:



```
Private Sub moApp_SheetChange(ByVal Sh As Object, ByVal Target As Range)

    ' Context: werkmap
    [applicatiespecifieke code]

    If Not goTool.Sheet Is Nothing Then

        If Sh.Parent Is goTool.Sheet.Parent Then
            ' Context: werkmap
            '[werkmapspecifieke code]
        End If

        If Sh Is goTool.Sheet Then
            ' Context: werkblad
            [Werkbladspecifieke code]
        End If
    End If
End Sub
```

Bij het op de gewenste wijze genereren en besturen van de opdrachtknoppen kan de code echter in omvang al snel exploderen worden. Vandaar dat hierna een oplossing wordt geboden:

4. Zoveel mogelijk zoveel (context-afhankelijke) code centraliseren en hergebruiken

Uiteraard wordt indien mogelijk eerst zoveel mogelijk code gecentraliseerd:

```
Private Sub moApp_WorkbookActivate(ByVal Wb As Workbook)
    ' Context: applicatie
    Application.StatusBar = "Workbook " & Wb.Name & " is actief"
    ' Context: werkmap
    HandleContextWorkbook
    ' Context: werkblad
    HandleContextWorksheet
End Sub
```

Een voorbeeld van centrale afhandeling ziet er zo uit:

```
Private Sub HandleContextWorkbook()

    Dim bEnabled As Boolean

    With goTool

        If Not .Sheet Is Nothing Then
            bEnabled = IIf(ActiveWorkbook Is .Sheet.Parent, True, False)
        End If

        Call .MenuBarCtrl.ICB_EnableControl(TAG_REFRESH, bEnabled)
        Call .MenuBarCtrl.ICB_EnableControl(TAG_SUBMENU, bEnabled)

        Call .CommandBar.ICB_EnableControl(TAG_REFRESH, bEnabled)
        Call .CommandBar.ICB_EnableControl(TAG_SUBMENU, bEnabled)
    End With
End Sub
```

Op deze wijze kan voor elke context een routine worden gecreëerd. Het zal echter niet altijd zinvol zijn om controls centraal aan te sturen. Bijvoorbeeld als het om slechts één control gaat:

```
Private Sub moApp_WorkbookBeforeSave(ByVal Wb As Workbook, ByVal SaveAsUI As Boolean, Cancel As Boolean)

    If Not goTool.Sheet Is Nothing Then
        If Wb Is goTool.Sheet.Parent Then
            ' Context: werkmap
            Call goTool.CommandBar.ICB_EnableControl(TAG_EXPORT, False)
        End If
    End If
End Sub
```



Tot slot:

Voor het toevoegen van één customized control aan een CommandBar is ten minste de volgende code nodig:

```
With cbc.Controls.Add(Type:=msoControlButton)
    .Caption = "&Exporteren..."
    .Tag = TAG_EXPORT
    .OnAction = "ThisWorkbook.ShowTag"
    .FaceId = 681
End With
```

De toepassingscontrols zullen binnen de twee opdrachtbalken meestal hetzelfde zijn. Ze moeten echter wel per opdrachtbalk worden toegevoegd. Een manier om in dit geval code te minimaliseren is door de controls te kopiëren: de controls worden vanuit het submenu in de menubalk gekopieerd naar de werkbalk. Bijkomend voordeel hierbij is dat per control kan worden bepaald of de control moet worden toegevoegd aan de werkbalk.

Controls van het type `msoControlButton` kunnen echter niet zomaar worden gekopieerd daar in dat geval de `Caption` op de knop wordt getoond in plaats van het gewenste `FaceID`. Daarom wordt van de volgende constructie gebruik gemaakt:

```
Dim cb As CommandBar
Dim cbcSource As CommandBarControl
Dim cbcTarget As CommandBarControl

' Creeer een floating werkbalk
Set cb = Application.CommandBars.Add(Name:=TOOL_NAME, Position:=msoBarFloating, _
    MenuBar:=False, Temporary:=False)

' Neem de controls over van de menubalk
For Each cbcSource In CommandBars(MenuBar).Controls(MENUBAR_CTRL).Controls

    Select Case cbcSource.Type

        Case msoControlPopup, msoControlComboBox
            ' Kopiëer dit type control van een menubalk-control naar een werkbalk;
            ' Een faceID bestaat niet en de caption en eventuele sub-items worden gekopieerd.
            Set cbcTarget = cbcSource.Copy(cb)
            cbcTarget.BeginGroup = cbcSource.BeginGroup
        Case msoControlButton

            Select Case cbcSource.Tag
                Case TAG_CLOSE
                    ' Doe niets. Control wordt niet getoond in onze werkbalk
                Case Else
                    ' Neem eigenschappen over van het menubalk-control
                    With cb.Controls.Add(Type:=msoControlButton)
                        .Caption = cbcSource.Caption
                        .ToolTipText = cbcSource.Caption ' Voeg de tooltipText toe
                        .BeginGroup = cbcSource.BeginGroup
                        .Tag = cbcSource.Tag
                        .OnAction = cbcSource.OnAction
                        .FaceId = cbcSource.FaceId
                        .Enabled = cbcSource.Enabled
                        .Visible = cbcSource.Visible
                    End With
                End Select

            Case Else
                Err.Raise 666, , "CommandBarControltype is niet geïmplementeerd!"
            End Select
        End Select
Next
```

Op deze manier is creatie van de individuele opdrachtknoppen op één plek opgelost.



Conclusie

Het is mogelijk om contextueel gedrag van CommandBar-controls onderhoudsvriendelijk en eenduidig te regelen. Voorwaarde is wel dat aan de basale programmeeruitgangspunten wordt voldaan. Hiermee wordt met name bedoeld het opzetten van logisch objectenmodel en het zoveel mogelijk centraliseren van code. Het gebruik van opdrachtbalk-wrapper-classes die een specifieke interface implementeren zorgt hierbij voor veel flexibiliteit.

Uiteindelijk zul je altijd goed van te voren moeten bedenken wat welke control na welk event in welke context moet gaan doen. Het spreekt vanzelf dat grondig testen daarbij een absolute vereiste is. Voor maatwerktoepassingen bestaan nou eenmaal geen standaardoplossingen.

Ik hoop echter dat dit whitepaper voldoende handvatten biedt om deze complexe materie op een robuuste en gebruikersvriendelijke manier op te zetten en te onderhouden.

Opmerkingen.

- Het Excel-bestand Commandbars.xla bevat de functionaliteit zoals in dit artikel uiteen is gezet. Het kan goed als template worden gebruikt voor andere maatwerkfunctionaliteit met betrekking tot opdrachtbalken.
- In dit bestand wordt ook gedemonstreerd hoe om te gaan met toepassingsopdrachtbalken wanneer het niet om een addin gaat.
- Het Excel-bestand bevat tevens dummy-code om specifieke voorbeelden te implementeren. Dit wordt als commentaar in de code aangegeven.

Emiel Nijhuis
Delegate
<http://www.delegate.nl>