Access voor Beginners - Hoofdstuk 26

Handleiding van Helpmij.nl

Auteur: OctaFish

Oktober 2014

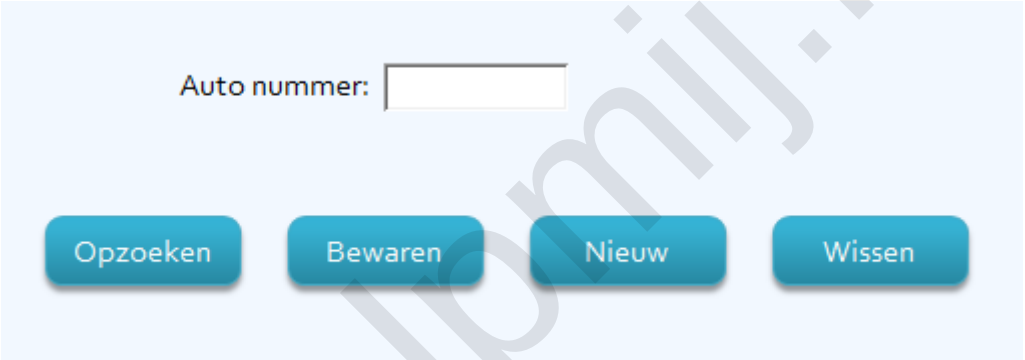
Werken met Klassenmodules (2)

In het vorige hoofdstuk hebben we een begin gemaakt met klassenmodules. We hebben gezien hoe je een klassenmodule aanmaakt, en we hebben de begrippen Property Let en Property Get behandeld. Tevens hebben we gezien dat je aan objecten gebeurtenissen kan hangen, zoals Opslaan, Verwijderen en Bijwerken.

In deze aflevering gaan we de klassenmodule gebruiken op een formulier. Dat formulier zal in eerste instantie nog niet zoveel doen, omdat we met kleine stappen willen laten zien wat er allemaal gebeurt met de verschillende objecten in de klassenmodule. Het gaat er voorlopig even om dat je een goed begrip krijgt van hoe e.e.a. allemaal (samen)werkt. De meeste opdrachten in dit hoofdstuk kun je waarschijnlijk veel simpeler oplossen met gewone variabelen, maar daar moet je dus even doorheen kijken!

Het formulier inrichten

We beginnen met het maken van een simpel formulier. Dat bevat een aantal knoppen, en een tekstvak. Bijvoorbeeld iets als:



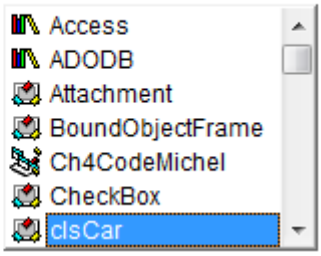
We beginnen, als we het tekstvak en de knoppen hebben gemaakt, weer met het declareren van een aantal variabelen.

```
(Algemeen)
Option Compare Database
Dim WithEvents objCar As clsCar
Dim strTestPropertyVal As String
```

De onbekende in dit plaatje is natuurlijk de variabele objCar. Deze variabele verwijst naar de *klassenmodule* die we in het vorige hoofdstuk hebben gemaakt. Een klassenmodule kan meerdere malen gebruikt worden, en zelfs in dezelfde procedure. Daarvoor gebruik je dan steeds andere variabelen met een (uiteraard) unieke naam. Een techniek die je bijvoorbeeld ook gebruikt als je meerdere Recordsets tegelijk declareert.

Doordat Access IntelliSense gebruikt (je weet wel: je typt een commando en een punt of spatie, en Access vult de keuzelijst met beschikbare opties) kun je de klassenmodule bij het declareren gewoon uit de keuzelijst halen.

```
Option Compare Database
Dim WithEvents objCar as
```



Verder zie je in de declaratie nog de tekst *WithEvents* en die geeft aan dat er met deze variabele gebeurtenissen (Events) kunnen worden uitgevoerd. En die gaan we dan ook straks definiëren.

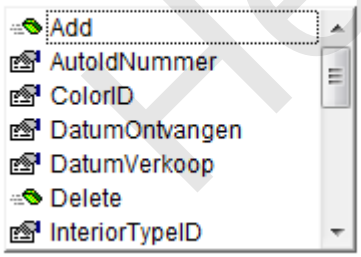
Het declareren deden we in de algemene sectie van het formulier, maar het *gebruik* van de klasse moeten we op een andere plek initialiseren. De meest logische plaats is de procedure <Bij laden> van het formulier, want deze actie hoeft maar één keer plaats te vinden, en de OnLoad gebeurtenis gebeurt maar één keer.

```
Private Sub Form_Load()
    Set objCar = New clsCar
End Sub
```

De volgende gebeurtenis die we gaan maken is voor de knop <Opzoeken>. In mijn voorbeeld heet de knop <Opzoeken> cmdLookup, en we gebruiken de gebeurtenis <Bij klikken> om de actie uit te voeren.

We maken uiteraard weer gebruik van de IntelliSense-functie van Access. Bij het declareren van de variabele objCar zagen we dat de klassenmodule in de uitklaplijst stond. We zien hetzelfde bij intypen van de code achter objCar; zodra je een punt typt, zie je de in de klassenmodule vastgelegde eigenschappen en methodes verschijnen.

```
With objCar
    .
```



En dat is natuurlijk hartstikke handig! Want niet alleen hoef je nu niet meer te onthouden welke eigenschappen en methodes er zijn, je kunt ook geen typefout meer maken. De code ziet er dan zo uit:

```
Private Sub cmdLookup_Click()
    With objCar
        .AutoIdNummer = Me.txtAutoID
        .Retrieve (.AutoIdNummer)
    End With
End Sub
```

Door met With .. End With te werken, kun je alle subeigenschappen (in dit geval van objCar) al kiezen door een punt te typen. Dat houdt de code korter en overzichtelijker.

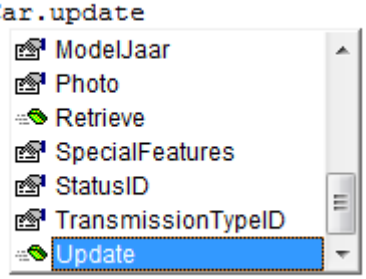
We vullen de property AutoIDNummer met de waarde die we op het formulier in het tekstvak txtAutoID hebben ingevuld. Vervolgens starten we de property Retrieve om de waarde te verwerken.

```
Public Function Retrieve(ByVal AutoIdNummer) As ADODB.Recordset
    Debug.Print "clsCar.Retrieve methode"
    Debug.Print "Auto ID: " & AutoIdNummer & " is opgehaald"
End Function
```

Op dit moment doen we nog niet veel mee met de functie, zoals ik in het vorige hoofdstuk al heb uitgelegd. We genereren een tekst in het Venster Direct (te openen met de sneltoets <Ctrl>+<g> of via het menu <Beeld>, <Venster Direct>). Tekst in het venster Direct maak je aan met de opdracht Debug.Print gevolgd door de tekst die je wilt zien. De opdracht genereert zelf een nieuwe regel, dus die hoef je meestal niet mee te geven.

Voor de knop <Opslaan> gebruiken we de methode Update die we hebben aangemaakt.

```
Private Sub cmdSave_Click()
    objCar.update
End Sub
```



De methode Update heb je in het vorige hoofdstuk ingeklopt, maar we hebben hem nog niet echt besproken. De code ziet er als volgt uit:

```
Public Sub Update()
    Debug.Print "clsCar.Update methode"
    Debug.Print "Record voor AutoIdNummer: " _
        & Me.AutoIdNummer & " is bijgewerkt."
    RaiseEvent ActionSuccess("Updates voor bestaand " _
        & "record successvol bewaard!")
End Sub
```

Ook met deze functie doen we nog niet zo veel. Eigenlijk doen we twee keer hetzelfde, namelijk een melding genereren. De laatste functieaanroep in de procedure Update is ActionSuccess, die we als parameter een tekststring meegeven die we willen laten zien.

De procedure ActionSuccess is dus relatief simpel: hij genereert een tekst die je kunt laten zien aan de gebruiker.

```
Private Sub objCar_ActionSuccess(strMessage As String)
    MsgBox strMessage
End Sub
```

Meerdere exemplaren van een klassenmodule gebruiken

Voor het gebruik van een klassenmodule heb je hierboven gezien dat deze moet worden toegewezen aan een variabele. Wil je meerdere exemplaren tegelijk gebruiken, dan volstaat het om daar aparte variabelen voor te declareren. Je krijgt dan iets als:

```
Dim objCar1 As clsCar
Dim objCar2 As clsCar
Dim objCar3 As clsCar

    Set objCar1 = New clsCar
    Set objCar2 = New clsCar
    Set objCar3 = New clsCar
```

En daarna kun je aan elk object weer eigenschappen veranderen etc.

Werken met Nummeringsvariabelen

In het vorige hoofdstuk stond een variabele gedeclareerd die ik nog niet eerder heb gebruikt. Het wordt tijd om de werking daarvan te uit te leggen!

Een nummeringsvariabele is, zoals de naam al zegt, een *variabele*. Het is een variabele die, net als de variabele Const, vaste waarden bevat. Deze waarden worden bij het declareren al vastgelegd, en kun je daarna oproepen.

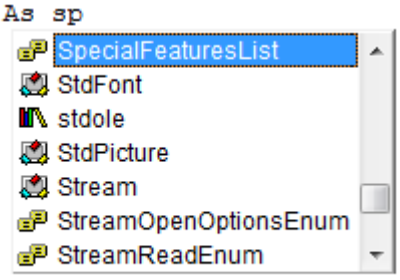
In het voorbeeld maakten we een nummeringsvariabele die een lijst met verschillende opties bevat. Hij wordt als volgt gemaakt:

```
Public Enum SpecialFeaturesList
    Zonnedak
    Schuifdak
    Boordcomputer
    NavigatieSysteem
    TelefoonAansluiting
    USBAansluiting
    Overig
End Enum
```

De variabele wordt *Public* gedeclareerd, zodat we hem overal kunnen gebruiken. Met *Enum* begin je de lijst, en met *End Enum* wordt hij afgesloten. Daar tussen typ je de opties die je in de lijst wilt zien.

Net als bij de klassenmodule, is de nummeringsvariabele vervolgens weer met IntelliSense op te zoeken:

```
Public Property Get SpecialFeatures() As sp
```




```
End Property
```

In dit geval hoef je maar twee letters te typen om hem te vinden.

En als je de Property *SpecialFeatures* gebruikt, is de keuze beperkt tot de waarden die in de lijst staan.

```
objcar.SpecialFeatures =
```

Dat maakt het werken met de eigenschap natuurlijk een stuk eenvoudiger. Merk op dat de opzoeklijst die je ziet niet helemaal gelijk is aan de lijst zoals hij is ingetypt. De keuzelijst is namelijk gesorteerd op alfabet zodat je makkelijk kunt zoeken door één of twee letters te typen. Wil je dat de oorspronkelijke sortering gebruikt wordt, dan zou ik de lijst nummeren. Dat ontneemt de lijst dan wel weer de mogelijkheid om de eerste letter in te typen, want je zult dan een cijfer moeten typen. Dus of die oplossing echt handig is, mag je zelf bepalen!

Properties toevoegen aan bestaande objecten

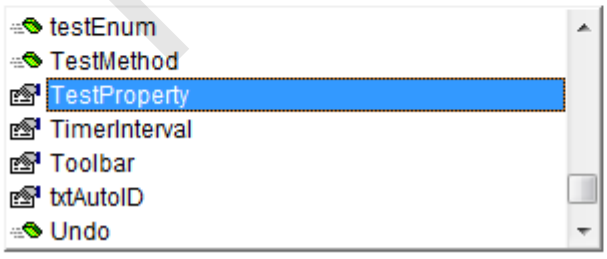
We hebben tot nu toe properties gemaakt in een klassenmodule, maar je kunt ook properties maken die je binnen een bestaand object gebruikt. Ze worden dan toegevoegd aan de eigenschappen van dat object. Denk bijvoorbeeld aan de eigenschappen van een formulier waar je eigen properties aan toevoegt. Een voorbeeldje ziet er zo uit:

```
Option Compare Database
Dim WithEvents objcar As clsCar
Dim strTestPropertyVal As String

Public Property Get TestProperty() As String
    TestProperty = strTestPropertyVal
End Property

Public Property Let TestProperty(ByVal Value As String)
    strTestPropertyVal = Value
End Property

Sub ShowTestProperty()
    me.test
```




```
End Sub
```

In het formulier frmAutoDetails heb ik eerst een variabele strTestPropertyVal gedeclareerd. Vervolgens is op het formulier een *Property Get* gemaakt met de naam TestProperty, en een *Property Let* met dezelfde naam. Om de nieuwe eigenschap te testen, maak je een nieuwe procedure, en in die procedure typ je **Me.test** waarna de property TestProperty tevoorschijn komt in de lijst.

Boven de property *TestProperty* zie je de methode *TestMethod* staan. Ook deze functie is toegevoegd aan het formulier. De code daarvan ziet er zo uit:

```
Public Sub TestMethod()
    MsgBox "Deze methode is toegevoegd aan het formulier"
End Sub
```

Je kunt een Property of Method overigens herkennen aan het icoontje dat voor de naam staat. Een Property is, zoals we nu weten, een eigenschap en die worden altijd aangeduid met een handje en een kaart. Een methode wordt weergegeven door een groep vliegend boekje (of wat je er ook in wilt zien…)

Samenvatting

In dit hoofdstuk zijn we iets dieper ingegaan op de manier waarop we eigen *Properties* en *Methods* kunnen gebruiken. We hebben ook gezien dat we variabelen kunnen declareren met een vaste lijst, die we vervolgens kunnen opvragen in een property. Als laatste hebben we gekeken naar properties die we kunnen toevoegen aan bestaande objecten, zoals formulieren.

Volgende hoofdstuk

In het volgende hoofdstuk gaan we het formulier koppelen aan de database, zodat we gegevens kunnen toevoegen en muteren. Daarbij maken we volop gebruik van de hoofdstukken waarin je hebt geleerd hoe je recordsets moet maken i.c.m. niet-gebonden formulieren.