Hoofdstuk 3

Gestructureerde VBA-programma's schrijven

In dit hoofdstuk:

- De structuur die de meeste programma's gebruiken
- Profiteren van structuur
- Programma's maken met de macrorecorder
- Een programma maken met behulp van een Sub
- Een programma maken met behulp van een Function
- Bepaalde programmaonderdelen verbergen door middel van bereik
- Witruimte aan programma's toevoegen
- Commentaar aan de code toevoegen

In de hoofdstukken 1 en 2 heb ik de grondbeginselen van programmeren in VBA besproken zonder me te bekommeren om structuur, een essentieel kenmerk van VBA-programma's. Door structuur toe te voegen maak je de code leesbaarder en gemakkelijker te gebruiken. Het is bovendien een verplicht onderdeel van het ontwikkelingsproces.

Je kunt op verschillende manieren naar de structuur van een programma kijken. Net zoals je in een presentatie een aantal onderdelen puntsgewijs opsomt, zo moet je een programma structuur geven om het goed te laten werken. Structuur in een programma kun je vergelijken met een paklijst voor een grote bestelling van een klant; het helpt je te verzekeren dat alles aanwezig is. De structuur die je aan een programma geeft, is vergelijkbaar met getallen in een grafiek: het maakt de inhoud van het programma inzichtelijker en begrijpelijker.

In dit hoofdstuk behandel ik verscheidene vormen van structuur. Het meest voor de hand liggende structurerende onderdeel is fysieke structuur. Fysieke structuur aanbrengen houdt in dat je een programma onderverdeelt in kleine stukken die je gemakkelijk kunt schrijven en die gemakkelijk te begrijpen zijn. Veel mensen verdelen programma's in taakspecifieke stukken. Wat dat betreft is het leerzaam om de macrorecorder te gebruiken, want dan zie je hoe je grote procedures in kleinere taken kunt opsplitsen. Ik zal daarom eerst aangeven hoe je de macrorecorder gebruikt om meer over je specifieke programmabehoeften te weten te komen.

Een andere vorm van structuur heeft betrekking op privacy. Het is belangrijk om rekening te houden met wie jouw programma kunnen zien en tot in hoeverre ze het programma kunnen gebruiken. *Bereik*, oftewel de programmatoegang bepalen, is belangrijk omdat je sommige delen geheel privé en andere delen geheel openbaar zult willen maken.

Ten derde zijn er visuele structuuronderdelen. Hoe je witruimte bij het schrijven van een programma gebruikt, kan het verschil maken tussen leesbaarheid en een warboel. In hoofdstuk 2 bespreek ik het gebruik van commentaarregels in de vorm van pseudocode, maar je zult wellicht merken dat er nog meer commentaar nodig is om iemand het programma echt te helpen begrijpen (en zelf het overzicht te behouden wanneer je later wijzigingen moet aanbrengen).

Onderdelen van een programma

In de voorbeelden van hoofdstuk 1 en 2 heb ik fysieke structuur gebruikt. Je kunt geen programma schrijven, zelfs niet een simpel programma, zonder in elk geval een beetje structuur aan te brengen, aangezien VBA die structuur nodig heeft om te begrijpen wat je wilt doen. Een VBA-gebruiker kan de structuur negeren, maar VBA niet. In dit hoofdstuk zullen we de betekenis van elk structureel onderdeel bezien.

De onderdelen van een programma definiëren

Een programma is het hoogste niveau van fysieke structuur. Een programma bevat alles wat nodig is om een bepaalde taak te realiseren. Een programma kan de grenzen van modules, klassenmodules en formulieren overschrijden. (Modules, klassenmodules en formulieren zijn speciale containers die programmacode bevatten. Jij kunt ze als afzonderlijke bestanden opslaan voor later gebruik, maar Office sluit ze in een toepassingsdocument of -sjabloon in.)

Het begrip *programma* dateert al uit de oertijd van computers. Een programma is een verzameling code waarmee functies worden geïmplementeerd die door het besturingssysteem of de gebruiker worden verlangd.

Sommige mensen hebben moeite om te begrijpen wat een programma is, omdat moderne softwarepakketten de term vaak onjuist definiëren. Wanneer je Microsoft Word opent, gebruik je een programma. Microsoft Office daarentegen is een verzameling toepassingen, een *suite* genoemd. Het is niet één programma, maar een hele serie programma's, waaronder Word, PowerPoint en Excel. Ook wanneer je Kladblok opent, open je een programma. Kladblok en Word bieden elk een verzameling functies die de gebruiker wenst om een bepaald taak te verrichten.

Andere programma's zijn er ten behoeve van het besturingssysteem. Een driver ofwel apparaatstuurprogramma (bijvoorbeeld het stuurprogramma van je muis) is een programma. Ook de gebruikersinterface die

je de muis helpt configureren is vaak een afzonderlijk programma. Het is niet een onderdeel van het apparaatstuurprogramma, ook al regelt het de handelingen van dat stuurprogramma. Wanneer je Excel vanuit Word aanroept, versmelten de twee programma's niet tot een eenheid, maar gebruik je nog steeds twee programma's om één taak te verrichten.

Een programma is niet hetzelfde als een project. Een *project* is de container die de modules, klassenmodules en formulieren bevat die bij een bepaald document horen. Je maakt niet een nieuw programma wanneer je een nieuw project maakt. Een VBA-project kan in feite een heleboel VBA-programma's bevatten.

Elke openbare subprocedure waarmee een gebruiker in het dialoogvenster Macro kan werken (zie hoofdstuk 2) is een afzonderlijk programma. Het programma ZegHallo, dat ik in hoofdstuk 2 bespreek, is een voorbeeld van een eenvoudig programma dat van een enkele subprocedure gebruikmaakt, maar je kunt programma's zo ingewikkeld maken als je wilt.

In hoofdstuk 16 laat ik zien dat een programma projectgrenzen kan doorbreken. Alle voorbeelden in dat hoofdstuk maken gebruik van de diensten van minstens twee Microsoft Office-producten om één taak uit te voeren. Maar hoewel het programma de diensten inroept van twee of meer Microsoft Office-producten, is het nog steeds één programma. Elk voorbeeld bevat een verzameling functies die de gebruiker wenst om een bepaalde taak te verrichten. De fysieke locatie en het gebruik van externe modules veranderen niets aan die definitie.

Inzicht in de programmeerblokken van VBA

Een VBA-programma bestaat uit bouwstenen. In de praktijk geven mensen meestal fysieke voorbeelden om uit te leggen hoe de dingen werken, omdat programmeren een nogal abstracte bezigheid is. Toch moet je de abstracte onderdelen van VBA-programma's kennen, want anders kun je geen programma schrijven. In deze paragraaf worden de elementaire onderdelen van VBA-programma's besproken. In de paragraaf 'Leve de legoaanpak' verderop in dit hoofdstuk zal ik deze abstracte onderdelen aan de hand van een fysiek voorbeeld uitvoerig toelichten. Hier beschrijf ik slechts de volgende vier onderdelen:

- **Project.** Het project fungeert als een container voor de modules, klassenmodules en formulieren van een bepaald bestand. In Word zie je dat minimaal drie projecten in de Visual Basic Editor zijn geladen: de sjabloon Normal, de documentsjabloon en het document. Excel-gebruikers zien gewoonlijk slechts één project bij het bestand dat ze hebben geopend.
- **Module, klassenmodule en formulier.** Deze drie onderdelen fungeren als containers voor belangrijke programmeeronderdelen zoals klassenbeschrijvingen en procedures. Een project kan verschei-

dene modules, klassenmodules en formulieren bevatten. Elk van deze onderdelen moet echter een unieke naam dragen.

- **Function en Sub.** De onderdelen Function en Sub bevatten regels code (instructies genoemd). Een Function retourneert een waarde aan de oproeper, maar een Sub niet. Microsoft Office biedt via de Sub toegang tot codefunctionaliteit, maar niet via de Function. Daarom moet je altijd via een Sub toegang geven tot je VBA-programma.
- **Instructie.** Veel mensen noemen een afzonderlijke regel code een instructie. De pseudocode in hoofdstuk 2 laat zien waarom dat zo is. Elke regel pseudocode is een uitspraak over wat de toepassing moet doen. Het voorbeeld in de paragraaf 'Stap 2: Het ontwerp implementeren' in hoofdstuk 2 laat zien hoe deze regels pseudocode worden vertaald in code die voor VBA begrijpelijk is. Een instructie is gewoon een uitspraak in VBA-taal.

Werken met de macrorecorder

Met de macrorecorder kun je toetsindrukken en handelingen van een VBA-programma opnemen. Je kunt er volledige taken mee opnemen, bijvoorbeeld een document instellen, maar ook gedeeltelijke taken, bijvoorbeeld tekst markeren en bepaalde opmaak geven. De macrorecorder kan je helpen bij de uitvoering van de volgende taken:

- een macro maken op grond van jouw handelingen;
- ontdekken hoe Word bepaalde taken uitvoert;
- beslissen hoe je jouw programma in taken opsplitst;
- een basis leggen voor een ingewikkelder programma.

De macrorecorder is niet de oplossing voor ál je VBA-bezigheden. Je kunt de macrorecorder bijvoorbeeld niet gebruiken om zonder extra code interactieve programma's te maken. Hetzelfde geldt voor programma's die op grond van invoer door de gebruiker, de omgeving of gegevens hun koers eventueel moeten wijzigen. Bij al deze taken is het nodig dat je code toevoegt. Ondanks alles vormt een opgenomen macro een goed vertrekpunt voor veel gestructureerde programmeertaken. Je krijgt de grondbeginselen met de macrorecorder snel onder de knie, waarna je zo nodig wijzigingen aanbrengt. Het opnemen van een macro verloopt op dezelfde elementaire manier, ongeacht welke versie van Office je gebruikt:

1. Start de macrorecorder.
2. Neem alle stappen die je normaal gesproken neemt om een taak uit te voeren.
3. Stop de macrorecorder.

4. Sla de macro op zodra de Office-toepassing dat aan je vraagt.

5. Open de macro eventueel en breng wijzigingen aan als dat nodig is.

Een macro opnemen via het lint

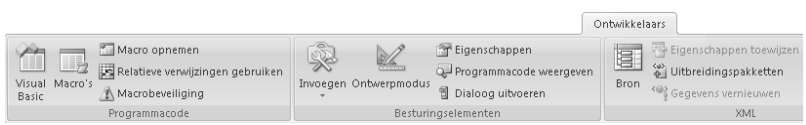
Een macro opnemen met de nieuwe functies die door de lintinterface van Office 2007 worden geboden, is gemakkelijker dan in de vorige versies van Office. Microsoft heeft functies toegevoegd die het eenvoudiger maken om een macro te maken. Als je bijvoorbeeld op Alt drukt, zie je in een vakje boven elk besturingselement van het lint het getal of de letter die je moet indrukken om een bepaalde actie uit te voeren.

Als je gewend bent om de meeste taken in Office met de muis uit te voeren, kun je de toetsen die je nodig hebt om de macro op te nemen, het beste uit je hoofd leren. Als je een macro opneemt zonder fouten wordt de macro namelijk sneller uitgevoerd en kun je de macro later gemakkelijker bewerken. De volgende stappen beschrijven hoe je een macro met het lint opneemt:

1. Plaats het tabblad Ontwikkelaars op de voorgrond.

   Zie de pagina 'Inclusief batterijen: VBA is met Office meegeleverd' in hoofdstuk 1, want dit tabblad wordt in Office 2007 niet standaard weergegeven.

   Je ziet nu het tabblad Ontwikkelaars (zie figuur 3.1)

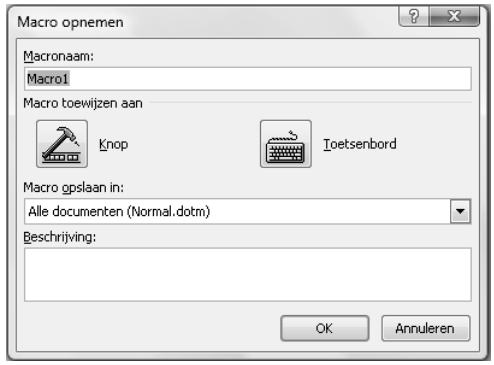


Figuur 3.1: Op het tabblad Ontwikkelaars tref je de meeste dingen aan die je nodig hebt om met macro's te werken

2. Klik op Macro opnemen.

   De Office-toepassing opent het dialoogvenster Macro opnemen, dat in figuur 3.2 is afgebeeld.

3. Geef de macro een duidelijke naam.



Figuur 3.2: In het dialoogvenster Macro opnemen geef je de macro een naam en eventueel een beschrijving

4. Typ een sneltoets voor de macro als je die via het toetsenbord wilt kunnen starten.

   Kies deze optie alleen voor de belangrijkste macro's, want anders raak je snel door de voorraad beschikbare sneltoetsen heen.

5. Selecteer de opslaglocatie in de vervolgkeuzelijst Macro opslaan in.

   De opslaglocatie is in elke Office-toepassing verschillend. Dit zijn de locaties in Excel:

   - **Dit werkboek.** Kies deze optie als je de macro binnen het lokale bestand wilt opslaan. Iedereen die het bestand opent, kan de macro bereiken.
   - **Persoonlijk Macrowerkboek.** Kies deze optie als je de macro wilt opslaan in een speciaal werkboek dat al je persoonlijke macro's bevat. Jij kunt de macro te allen tijde op deze opslaglocatie bereiken. Het maakt daarbij niet uit welk werkboek je opent.
   - **Nieuw werkboek.** Kies deze optie als je de macro in een nieuw werkboek wilt opslaan.

   De opslaglocaties in Word zijn vergelijkbaar, zoals uit het volgende lijstje blijkt:

   - **Document.** Kies deze optie als je de macro binnen het huidige bestand wilt opslaan. Iedereen die het bestand opent, kan de macro gebruiken.
   - **Documentsjabloon.** Kies deze optie als je de macro wilt opslaan binnen de sjabloon die voor het huidige document is gebruikt. Iedereen die een document maakt op basis van deze sjabloon, kan de macro gebruiken.

- **Alle documenten (Normal.dotm).** Kies deze optie als je de macro binnen de globale sjabloon wilt opslaan. Als je de macro hier opslaat, houdt dit in dat iedereen die een willekeurig document opent, de macro kan bereiken.

6. Typ een beschrijving van de macro in het invoervak Beschrijving.

   Het is essentieel dat je een volledige beschrijving typt, omdat dit de enige toelichting is die de macro na voltooiing bevat.

7. Klik op OK.

   De Office-toepassing begint de macro op te nemen. De knop Macro opnemen verandert in de knop Opname stoppen en de kleur van de knop verandert van rood in blauw.

8. Voer alle taken uit die je normaal zou verrichten om de taak te voltooien.

   De Office-toepassing registreert al je toetsindrukken. Je muisbewegingen worden echter niet opgenomen. Daarom moet je het gebruik van de muis vermijden en alle taken via het toetsenbord verrichten.

9. Klik op Opname stoppen.

   De Office-toepassing beëindigt de macro.

Je kunt de nieuwe macro bekijken door op het tabblad Ontwikkelaars op Macro's te klikken. Het dialoogvenster Macro toont de macro's die vanuit het huidige document kunt opvragen, ongeacht of ze in het huidige document of in een extern document of een sjabloon zijn opgeslagen. In de paragraaf 'Macro's wijzigen' verderop in dit hoofdstuk wordt dit dialoogvenster uitgebreider beschreven.

Een macro opnemen via het menu

In oudere versies van Office en in sommige Office 2007-producten moet je het menu gebruiken om de macrorecorder te activeren. Voer de volgende stappen uit om een macro via het menu op te nemen:

1. Kies Extra ⇨ Macro ⇨ Nieuwe macro opnemen.

   De Office-toepassing toont het dialoogvenster Macro opnemen (zie opnieuw figuur 3.2).

2. Typ een beschrijvende naam voor de macro.

3. Typ een toetsencombinatie voor de macro als je de macro via het toetsenbord wilt kunnen oproepen.

Kies deze optie uitsluitend bij belangrijke macro's, want anders raakt de voorraad beschikbare toetsencombinaties al snel uitgeput.

Sommige oudere Office-producten bieden nog meer opties. In oudere versies van Word kun je de macro niet alleen aan een toetsencombinatie koppelen, maar ook aan een werkbalkknop (of aan allebei) door op de gewenste knop te klikken.

4. Selecteer een opslaglocatie in de vervolgkeuzelijst Macro opslaan in.

   (Dit is een functie van Office 2007; oudere versies slaan de macro altijd op in het lokale document.)

   De opslaglocaties variëren per Office-toepassing. De locaties in Visio zijn:

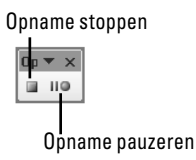
   - **Actieve document.** Kies deze optie als je de macro wilt opslaan binnen het lokale bestand. Iedereen die het bestand opent, kan de macro bereiken.
   - **Stencil.** Slaat het bestand op in het stencilbestand. Iedereen die het stencil gebruikt, kan de macro bereiken, ongeacht welk document werd geopend.

5. Typ een macrobeschrijving in het invoervak Beschrijving.

   Het is essentieel dat je een volledige beschrijving typt, omdat dit de enige opmerking is die de macro na voltooiing bevat.

6. Klik op OK.

   De Office-toepassing begint de macro op te nemen. Je ziet de werkbalk Opname stoppen (zie figuur 3.3). Deze werkbalk bevat opties om de opname van de macro te stoppen en te pauzeren. Door de macro te pauzeren kun je handelingen uitvoeren waarvan je niet wilt dat de macrorecorder ze opneemt. Klik nogmaals op Opname pauzeren om de opname voort te zetten.



Figuur 3.3: Stop of onderbreek zo nodig de opnameprocedure

7. Voer alle taken uit die je normaal zou verrichten om de taak te voltooien.

   De Office-toepassing registreert al je toetsindrukken. Muisbewegingen worden echter niet opgenomen. Daarom moet je het gebruik van de muis vermijden en alle taken via het toetsenbord verrichten.

8. Klik op Opname stoppen.

   De Office-toepassing beëindigt de macro. De werkbalk Opname stoppen verdwijnt van het scherm.

Je kunt je nieuwe macro bekijken door Extra ➪ Macro ➪ Macro's te kiezen. Het dialoogvenster Macro toont een lijst van de macro's die vanuit het huidige document kunt opvragen, hetzij lokaal, hetzij deel uitmakend van een ander document, een sjabloon of een ander gekoppeld bestand.

Macro's wijzigen

Een macro wijzigen met de macrorecorder verloopt niet veel anders dan een willekeurige andere macro wijzigen. Het enige verschil is dat je de oorspronkelijke code niet zelf hebt geschreven en dat de macrorecorder geen commentaarregels voor je invoegt. Open als voorbeeld de Excel-macro maar eens die je eerder in dit hoofdstuk hebt gemaakt in de paragraaf 'Een macro opnemen via het lint'. Voor deze taak voer je de volgende stappen uit:

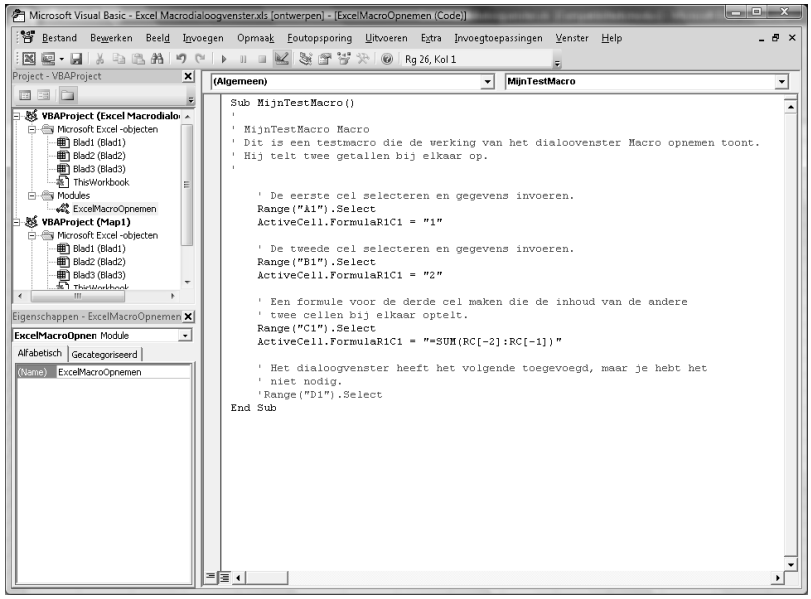
1. Open het dialoogvenster Macro. Als je Office met het lint gebruikt, klik je op het tabblad Ontwikkelaars op Macro's. Kies in oudere versies van Office Extra ➪ Macro ➪ Macro's.

   Je ziet het dialoogvenster Macro (zie figuur 3.4).

Figuur 3.4: Het dialoogvenster Macro toont een lijst van beschikbare macro's

2. Selecteer de macro die je wilt bewerken en klik daarna op Bewerken.

   De Office-toepassing opent de Visual Basic Editor met de geselecteerde macro geopend (zie figuur 3.5). Mogelijk zie je nog meer geopende macrobestanden.



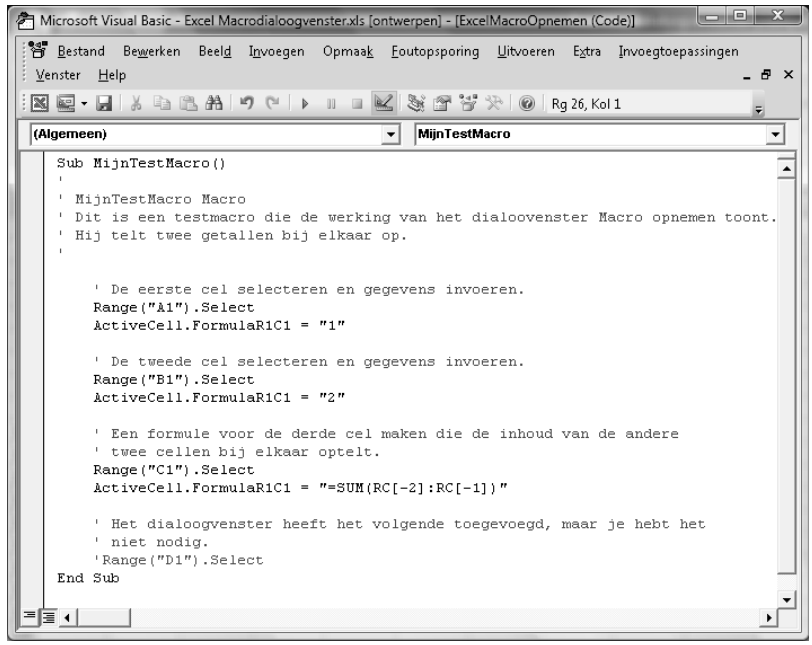
Figuur 3.5: In de Visual Basic Editor kun je de macro's wijzigen die je met de macrorecorder hebt gemaakt

3. Voeg commentaar toe aan de opgenomen macro, zodat je je stappen later nog kunt volgen.

4. Breng alle gewenste wijzigingen in de macro aan.

5. Sla de macro op en sluit de Visual Basic Editor.

Je hoeft de in figuur 3.5 afgebeelde macro op dit moment niet te begrijpen. Niettemin begint de macro met de toewijzing van de waarde 1 aan cel A1 van het werkblad. Omdat het invoegteken al in A1 stond toen de macro-opname startte, komt deze handeling niet in de macro voor. Deze afwezigheid is een van de redenen waarom je macro's die je met de macrorecorder opneemt zult willen bewerken. De macro verplaatst het invoegteken vervolgens naar cel B2 en geeft die cel de waarde 2. Tenslotte verplaatst de macro het invoegteken naar cel C3 en plaatst in die cel een formule die de twee getallen bij elkaar optelt. Je zou deze macro kunnen bewerken door de ontbrekende celverwijzing naar A1 toe te voegen, commentaar toe te voegen en de ene extra instructie te verwijderen, zoals je in figuur 3.6 ziet.



Figuur 3.6: Als je de macro voor verschillende taken wilt gebruiken, is het belangrijk om de uitvoer van de macrorecorder te bewerken

Gebruikmaken van subprocedures

In hoofdstuk 2 gebruik ik een subprocedure oftewel een Sub. Een Sub is de gemakkelijkste manier om code te verpakken en is de enige verpakkingsmethode die in het dialoogvenster Macro wordt weergegeven. Het voorbeeld in de paragraaf 'Je eerste functie schrijven' verderop in dit hoofdstuk demonstreert dit. Daarom is het belangrijkste entreepunt voor een programma de enige plek waar je altijd een Sub gebruikt, tenzij dat programma een hulpprogramma is dat je slechts voor programmeerdoeleinden gebruikt.

Een tweede geval waarin je een Sub gebruikt, is als je een taak uitvoert en niet een directe retourwaarde terugkrijgt. Je kunt een Sub inzetten om de gebruiker informatie te geven, zoals bij de berichten in hoofdstuk 2 het geval is. Een Sub kan informatie op een aantal manieren wijzigen, maar kan geen waarde retourneren; alleen een Function kan dat. Je kunt echter wel argumenten gebruiken als een van de manieren om informatie via een Sub te wijzigen (zie het voorbeeld in de paragraaf 'Je eerste functie schrijven'). Een tweede manier maakt gebruik van globale variabelen (zie het voorbeeld in de paragraaf 'De effecten van bereik bepalen' verderop in dit hoofdstuk)

Veel VBA-gebruikers gebruiken Subs ook als middel om code in stukken op te splitsen. In plaats van code die eindeloos voortgaat, kan code met Subs in paginagrote stukken worden opgesplitst. Dat maakt de code een stuk leesbaarder.

Werken met functies

Misschien zie je niet in wanneer je een Function zou moeten gebruiken nadat je enige tijd met Subs hebt gestoeid. Maar toch, niet elk probleem is een schroefje waarvoor je een schroevendraaier of een spijker waarvoor je een hamer nodig hebt. Je gebruikt een Function voor problemen die je met een Sub niet kunt oplossen. In de meeste gevallen is er een duidelijke reden om een Function dan wel een Sub te gebruiken. Je gebruikt altijd een Sub als je programmacode van binnen de hosttoepassing wilt bereiken, maar altijd een Function wanneer je een berekening met een geretourneerd resultaat wilt uitvoeren.

Een Function retourneert altijd een waarde, en dat maakt haar anders dan een Sub. Dat is de reden dat je functies kunt schrijven die code bevatten die je binnen een programma vaak wilt uitvoeren. Als je bijvoorbeeld een lijst namen wilt verwerken kun je een Function maken die elke naam afzonderlijk verwerkt, waarna je die Function eenmaal voor elke naam oproept. De Function kan de verwerkte informatie in de vorm van een retourwaarde overhandigen. In hoofdstuk 5 beschrijf ik hoe je repeterende code maakt met structuren zoals Do… Until.

Je kunt ook een Function gebruiken voor openbare code die je niet in het dialoogvenster Macro wilt vermelden. Gewoonlijk zie je in het dialoogvenster Macro geen Functions opgesomd staan, maar alleen Subs.

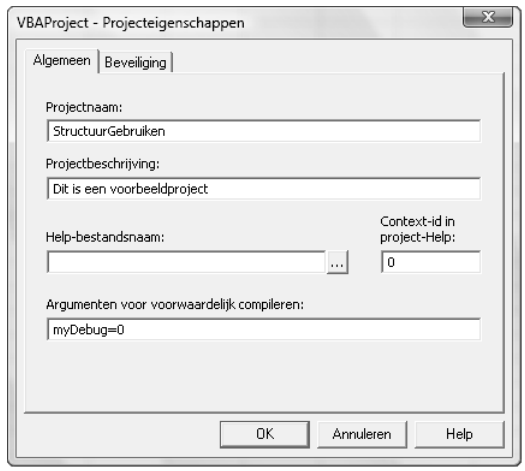
De projectinstellingen wijzigen

Tot dusverre heb je VBA gebruikt zonder al te veel opties te configureren. Ook de voorbeelden maakten gebruik van de standaardinstellingen die VBA normaal gesproken aanhoudt. De meeste VBA-programmaniveaus worden echter tot op zekere hoogte geconfigureerd, inclusief het project. In deze paragraaf beschrijf ik de diverse projectinstellingen.

Je opent de projectinstellingen door in de Projectverkenner met de rechtermuisknop op het project te klikken en in het snelmenu de optie Eigenschappen van project te kiezen. Je ziet dan het dialoogvenster Projecteigenschappen (zie figuur 3.7). Zie de paragraaf ‘Een blik op de geïntegreerde ontwikkelomgeving (IDE)’ in hoofdstuk 1 voor een beschrijving van de Projectverkenner. (In figuur 3.7 heb ik alvast de diverse opties ingevuld die ik in de volgende paragrafen zal bespreken.)

Je project beschrijven

Door een project te beschrijven zul je dat project gemakkelijker kunnen vinden wanneer je het in de vensters van de Visual Basic Editor bekijkt. Begin met het project een duidelijke naam te geven. Dat hoeft geen lange naam te zijn. Een naam als *VBAProject* is echter nietszeggend, omdat hij niet aangeeft wat het doel van het project is. Het invoervak Projectnaam (zie figuur 3.7) bevat een duidelijke projectnaam voor dit hoofdstuk omdat het voorbeeld demonstreert hoe je structuur aan je programma’s geeft.



Figuur 3.7: Verstrek elementaire informatie over je projecten

Hoewel de waarde van het vak Projectnaam op een aantal plaatsen wordt weergegeven, wordt de waarde van het invoervak Projectbeschrijving uitsluitend in de Object Browser getoond. Je kunt een langere tekstbeschrijving van het project schrijven, zodat je elk door de Object Browser getoond item gemakkelijker kunt bijhouden. Anders zou het wel eens lastig kunnen worden om het gewenste project te vinden.

Sommige mensen beschouwen Help-bestanden niet als een noodzakelijke projectbeschrijving, maar een *Help-bestand* is een gedetailleerde beschrijving van wat het project bewerkstelligt. Het vak Help-bestandsnaam geeft aan waar je het Help-bestand kunt vinden en welk Help-bestand je moet gebruiken. Het vak context-ID in project-Help bevat het getal van het Help-onderwerp dat op het project betrekking heeft. Dit item bevat gewoonlijk het onderwerpnummer van een overzichtspagina.

Voorwaardelijke compilatie toevoegen

Voorwaardelijke compilatie is essentieel als je meerdere versies van je programma wilt maken. Meestal doorloopt VBA de lijst van instructies en voert ze een voor een uit. Voorwaardelijke compilatie stelt VBA echter in staat om een taak op de ene manier uit te voeren terwijl je bezig bent met het programma te schrijven en te testen, en op een andere manier uit te voeren nadat je het programma hebt voltooid.

Deze functie wordt meestal gebruikt om een programma te helpen debuggen. In hoofdstuk 6 laat ik je zien hoe je een toepassing met voorwaardelijke compilatie foutenvrij maakt. Nu zullen we echter, gewoon voor de grap, een programmaatje maken dat laat zien hoe deze functie werkt. Let er om te beginnen op dat je in het invoervak Argumenten voor voorwaardelijke compileren `myDebug=0` typt en vervolgens op OK klikt. Typ daarna het volgende programma in een module. (Zie de paragraaf 'Stap 2: Het ontwerp implementeren' in hoofdstuk 2 als je wilt weten hoe je een module maakt.)

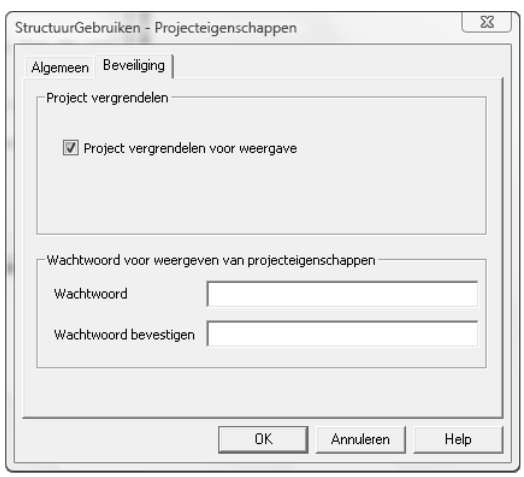
```
Public Sub CheckConditional()
    #If myDebug = 0 Then
        MsgBox "In standaardmodus"
    #Else
        MsgBox "In Debugmodus"
    #End If
End Sub
```

Dit programma zegt dat het programma een berichtvenster moet tonen met de tekst In standaardmodus als myDebug is ingesteld op 0. Als myDebug echter een willekeurige andere waarde heeft, moet een berichtvenster met de tekst In debugmodus worden getoond.

Open nogmaals het dialoogvenster Projecteigenschappen. Wijzig de inhoud van het veld Argumenten voor voorwaardelijke compileren zodanig dat er nu `myDebug=1` staat. Klik op OK en voer het programma nogmaals uit. Het programma toont nu een berichtvenster met de tekst In debugmodus.

Code vergrendelen

Op een gegeven moment kun je besluiten de code zodanig te vergrendelen dat niemand de code nog kan wijzigen. Op het tabblad Beveiliging van het dialoogvenster Projecteigenschappen (zie figuur 3.8) kun je dat doen. Markeer gewoon het selectievakje bij Project vergrendelen voor weergave en geef dan hetzelfde wachtwoord op in de invoervakken Wachtwoord en Wachtwoord bevestigen. Klik op OK om het proces te voltooien.



Figuur 3.8: Vergrendeling van je code kan bescherming bieden tegen spiedende blikken

Hoe veilig is vergrendelde code?

Een overweging bij het vergrendelen van code is dat het een vals gevoel van veiligheid kan geven. Hoewel codevergrendeling beginnende gebruikers ervan weerhoudt om aan je code te sleutelen, houdt het fervente krakers niet tegen. Op internet vind je veel fabrikanten die aanbieden om jouw Office-documenten voor je te ontgrendelen wanneer jij je wachtwoord vergeten bent. Helaas kan dezelfde toepassing die je helpt om je documenten op te halen wanneer je je wachtwoord vergeten bent, krakers ook toegang tot die documenten geven. Het beste wat je kunt doen is code vergrendelen als je beginnende gebruikers tegen zichzelf wilt beschermen, in plaats van vergrendeling te gebruiken als middel om je investering in de code zelf te beschermen.

Een project vergrendelen kan een goed idee lijken, maar je moet toch om een aantal redenen zorgvuldig overwegen of je dat gaat doen. De belangrijkste reden is dat je het project niet zonder het wachtwoord kunt ontgrendelen. Als je het wachtwoord vergeet, blijft het project misschien voor altijd op slot. De tweede reden is dat vergrendeling van een project iemand er niet van weerhoudt om je code te bekijken. Het enig wat vergrendeling van een project verhindert is dat iemand de code wijzigt. Om die reden biedt VBA niet de beste manier om je code te verbergen als dat om de een of andere reden nodig mocht zijn. (Je hebt in zo'n geval een echte codecompiler nodig, zoals Visual C++ of een product als Visual Studio Tools voor Office (VSTO), die codesoluties met Office-producten kan integreren.)

Compileropties kiezen

Wanneer je VBA voor het eerst start, neemt VBA aan dat je op een bepaalde manier code wilt schrijven. Deze veronderstellingen zijn niet altijd nauwkeurig, dus biedt Microsoft je een manier om VBA mee te delen dat er iets anders moet worden gedaan. De compileropties die in tabel 3.1 worden vermeld, helpen je aan te geven hoe VBA met jouw code moet omgaan. (Een compiler leest jouw code en vertaalt de woorden in instructies die Windows kan begrijpen.) Je kunt deze opties helemaal aan het begin van een module, klassenmodule of formulier toevoegen, voorafgaande aan alle andere code.

Tabel 3.1: Compileropties voor VBA

Optie	*Beschrijving*
Option Base <getal>	Kies deze optie als VBA een andere nummering van arrayonderdelen moet aanhouden. Je kunt arrayonderdelen nummeren vanaf 0 of 1. In hoofdstuk 9 lees je hoe je arrays gebruikt.
Option Explicit	Slimme VBA-programmeurs voegen deze compileroptie altijd aan hun code toe. Deze optie deelt VBA mee dat je variabelen wilt definiëren voordat je ze gebruikt. Deze optie maakt je code niet alleen beter leesbaar, maar kan je ook helpen om typefouten in je code op te sporen. Hoewel de standaardinstelling MijnVar en MijVar als twee variabelen behandelt, dwingt deze optie VBA om bij de foutief gespelde versie een vraagteken te zetten.
Option Compare <methode>	Kies deze optie als je de manier wilt wijzigen waarop VBA strings met elkaar vergelijkt. Wanneer je de Binary-methode kiest, beschouwt VBA ***Hallo*** en ***hallo*** als verschillende strings omdat de tweede string uitsluitend uit kleine letters bestaat. Als je de Text-methode kiest, betekent dit dat VBA ***Hallo*** en ***hallo*** als hetzelfde woord beschouwt omdat er niet op grote en kleine letters wordt gelet. De Database-methode, die alleen in Access beschikbaar is, hanteert de databasesorteervolgorde bij de vergelijking van strings.
Option Private Module	Kies deze optie als je een module privé wilt maken, zodat andere modules niet kunnen zien wat de module bevat. De termen public (openbaar) en private (privé) hebben betrekking op het ***bereik*** van het object. Verderop in dit hoofdstuk leg ik in de paragraaf 'Bereik binnen bereik' uit wat bereik inhoudt.

De instructie Option Explicit is zo belangrijk dat je haar eigenlijk altijd aan een programma moet toevoegen. Listing 3.1 geeft een beknopt voorbeeld van hoe deze functie werkt.

Listing 3.1: Option Explicit gebruiken om fouten te voorkomen

```
' Draag VBA op om te controleren dat we variabelen definiëren.
Option Explicit

' Deze Sub mislukt omdat de variabele niet wordt gedefinieerd.
Public Sub OptieControle()
    MijnVar = "Hallo"
    MsgBox MijnVar
End Sub

' Deze sub slaagt.
Public Sub OptieControle2()
    ' Definieer de variabele.
    Dim MijnVar As String
    ' Wijs een waarde toe aan de variabele.
    MijnVar ="Hallo"
    ' Toon een berichtvenster.
    MsgBox MijnVar
End Sub
```

In beide gevallen definieert de Sub een waarde voor een variabele met de naam MijnVar. De Sub OptieControle mislukt omdat de variabele niet wordt gedefinieerd voordat hij wordt gebruikt. VBA is zich niet bewust van MijnVar en kan deze variabele daarom niet gebruiken. Bekijk nu de tweede Sub, OptieControle2. Deze Sub werkt wel, omdat eerst de variabele MijnVar wordt gedefinieerd en er vervolgens een waarde aan wordt toegewezen. Hoewel het gebruik van de instructie Option Explicit wellicht een heleboel werk lijkt, bespaart deze instructie je echt tijd bij het opsporen van typefouten in je programma.

Leve de legoaanpak

Het is maar al te gemakkelijk om het schrijven van een programma nodeloos moeilijker te maken dan het is, doordat je in de abstracte aard van de code verstrikt raakt. De grootste fout die je kunt maken, is dat je één lang programma schrijft dat pagina's lang voortsuddert en dat door niemand wordt begrepen, zelfs niet door jou. Code die er meer uitziet als een doolhof dan als een eenvoudige verzameling instructies, wordt gewoonlijk *spaghetticode* genoemd.

De legoaanpak bij het schrijven van code houdt in dat de programmeur het programma opsplitst in gemakkelijk te begrijpen modules en vervolgens één module tegelijk gaat schrijven. Deze aanpak bevordert het schrijven van code die je naderhand gemakkelijk kunt wijzigen en die de meeste mensen gemakkelijk kunnen volgen. De legoaanpak maakt het bovendien gemakkelijker voor je om stukken code te verplaatsen wanneer je een ander programma schrijft.

Een toepassingsplan maken

In hoofdstuk 2 beschrijf ik hoe je pseudocode gebruikt om een programma te schrijven. Het programma in dat hoofdstuk bestaat slechts uit één module, aangezien het een heel eenvoudig programma is. Maar niet elk programma is zo simpel. Soms moeten er ingewikkelde taken worden uitgevoerd; in zo'n geval moet je eerst een overzicht of *toepassingsplan* maken. Om nogmaals met de legoaanpak te vergelijken: elk blok is een afzonderlijk stukje, maar je ziet pas het hele plaatje als je de blokken in elkaar past. Het toepassingsplan laat zien hoe de blokken gestapeld moeten worden om een bepaald resultaat te krijgen. Je hebt drie soorten blokken die je kunt gebruiken:

- projecten;
- modules, formulieren en klassenmodules;
- subprocedures en functies.

Wat is de legoaanpak?

Ik noem de werkwijze die ik in dit hoofdstuk beschrijf de legoaanpak omdat vrijwel iedereen dit speelgoed kent en het een beproefd middel is om goede programmeertechnieken uit te leggen. Anderen gebruiken andere namen voor deze manier van programmeren. Misschien heb jij de term modulair programmeren horen vallen. Daarmee wordt aangegeven dat men modules gebruikt, die afzonderlijke stukjes code bevatten. Verwar deze termen niet met objectgeoriënteerd programmeren, dat in hoofdstuk 8 wordt beschreven.

Bekijk het schrijven van een VBA-programma als volgt: je hebt een stapel blokken en in je hoofd een beeld van wat je wilt maken. Om je creatie te realiseren kies je blokken die eruitzien alsof ze passen. Je kunt standaardblokken kiezen uit dit boek of het VBA-hulpbestand of downloaden van websites, zoals FreeVBCode op http://www.freevbcode.com.

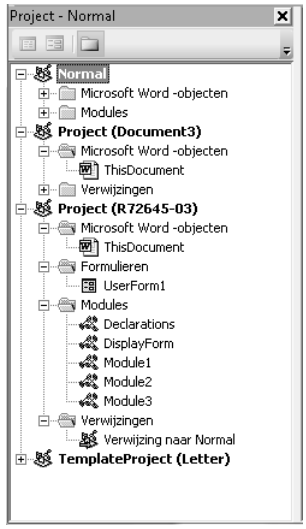
Je kunt je toepassingsplan op verschillende manieren opstellen. Ik begin gewoonlijk met een lijstje van de belangrijkste taken, zoals een rapport afdrukken of een woord zoeken. Dat is geen pseudocode. Je schrijft geen procedure die VBA moet volgen. Je bedenkt slechts wat de belangrijkste taken zijn die VBA moet uitvoeren. Misschien voert jouw programma slechts één taak uit. In dat geval kun je wellicht volstaan met één project, module en sub, zoals in het voorbeeld in hoofdstuk 2 het geval is.

Het project definiëren

Soms hoef je nooit meer dan een project voor een programma te definieren. Dat geldt vooral voor Excel- en Access-projecten, waar alle gegevenscontainers in de meeste gevallen in één bestand worden opgeslagen. Je kunt ook alle benodigdheden in één enkel Word-sjabloon plaatsen als dit de enige sjabloon is die je nodig hebt en je het programma voor verscheidene documenten kunt gebruiken.

Laten we even bij Word blijven om een aspect van projecten te bekijken. In figuur 3.9 zie je een kenmerkend venster van de Projectverkenner. Het venster bevat vier projecten: Normal, Project (Document3), Project (R72645-03) en TemplateProject (Letter).

Word laadt bovendien de sjabloon Normal. (Deze sjabloon kan verschillende extensies hebben, waaronder DOT, DOTX en DOTM in Office 2007.) Oudere versies van Office gebruiken altijd de extensie DOT. Word 2007 kan ook overweg met het nieuwe XML-formaat (DOTX) en het macro-ingeschakelde formaat (DOTM) van Microsoft. Als je macro's voor Word 2007 wilt maken, kun met het oog op de compatibiliteit het beste het DOT-bestand gebruiken, maar het DOTM-bestand biedt de grootste flexibiliteit en de meeste functies. Het is niet moegelijk om in een DOTX-bestand macro's op te slaan.



Figuur 3.9: Een programma kan verscheidene projecten omvatten

Je kunt ook een aangepaste sjabloon gebruiken. Aangepaste sjablonen bevatten speciale opmaak en macro's voor een bepaald documenttype, bijvoorbeeld brieven. Net als de sjabloon Normal hebben aangepaste sjablonen de extensie DOT, DOTX of DOTM. In een DOTX-bestand kunnen nooit macro's worden opgeslagen.

Ten slotte is er nog het document, dat ook als een project telt. Een document kan de extensie DOC, DOCX of DOCM hebben. In dit geval schreef ik een programma voor het afzonderlijke document, dat van sjabloonspecifieke Subs en Functions in de sjabloon Letter en van algemene Subs en Functions in de sjabloon Normal gebruikmaakt. In hoofdstuk 13 kun je dit voorbeeld in actie zien. Mogelijk verzeil jij ook wel eens in dit soort situaties wanneer je Word-programma's maakt.

Een andere situatie waarin je verscheidene projecten gebruikt, is wanneer je programma's maakt waarbij diverse toepassingen met elkaar moeten samenwerken. Ik heb bijvoorbeeld een programma voor Word dat aan CorelDRAW vraagt om een tekening te importeren en te converteren naar een formaat dat Word kan gebruiken. Word neemt de geconverteerde tekening in ontvangst en plaatst deze in het huidige document. Vervolgens vraagt Word aan Access om de tekening in een database op te zoeken. Access voert deze taak uit en retourneert een beschrijving van de tekening naar Word, waarna de beschrijving vervolgens wordt opgemaakt en in het huidige document onder de tekening wordt geplaatst. Als ik dit karweitje steeds zelf zou moeten uitvoeren wanneer ik een tekening nodig had, zou me dat per tekening twintig minuten kosten. Mijn VBA-programma heeft voor dit karweitje minder dan een minuut nodig en de resultaten zijn elke keer perfect.

Een programma definiëren houdt in dat je uitzoekt hoeveel bouwstenen je nodig hebt om een karwei te klaren. Je kunt de eenvoudige aanpak

kiezen en slechts één project voor veel programma's gebruiken. Maar maak je leven niet nodeloos moeilijk. Als je verscheidene projecten moet gebruiken, maakt VBA het je gemakkelijk.

Een module toevoegen

In de meeste beginprogramma's wordt slechts één module gebruikt. Een goed ontwerp begint echter met de definitie van modules. In principe zou je een reusachtige module kunnen bouwen, een hulpprogramma dat elk hulpprogramma bevat dat je ooit hebt geschreven, maar wat een warboel zou dat worden! Je kunt het vergelijken met een overvol dressoir waarin je moet zoeken naar de dasspeld of de armband die je vanavond wilt dragen: pas nadat je de hele inhoud van het dressoir hebt omgekieperd, vind je uiteindelijk waarnaar je op zoek was. Probeer in plaats van een enkele hulpmodule een module voor schijfactiviteiten en een andere module voor gegevensmanipulatie te maken. Wees uniek! Probeer je modules te organiseren en ga na wat het beste werkt.

Zodra je programmeervaardigheden zijn toegenomen, zul je op een of andere manier met de gebruiker willen interacteren. Je wilt de gebruiker bijvoorbeeld vragen stellen. Hiervoor zul je een formulier aan je programma moeten toevoegen. Voor elk formulier dat je wilt maken moet je een afzonderlijk formulier aan het programma toevoegen. Zorg ervoor dat je formulieren maakt die de gebruikers om één type informatie vragen; breng ze niet in verwarring door een stel ongerelateerde vragen op één formulier te stellen. (Zie hoofdstuk 7 voor een beschrijving van formulieren en hoe je ze gebruikt.)

Werken met objecten houdt in dat je klassen gebruikt. Je maakt een speciaal object door een klassenmodule aan je toepassing toe te voegen. Net als bij formulieren kun je het beste één klassenmodule toevoegen voor elk nieuw object dat je wilt maken. Zorg ervoor dat elk object dat je maakt, slechts één taak verricht. (In hoofdstuk 8 lees je hoe je met objecten werkt.)

Procedures ontwerpen

De meeste voorbeelden in dit boek maken gebruik van één project en één module. Mogelijk merk jij ook dat de meeste (zo niet alle) van je programma's eveneens één van deze blokken gebruiken. Bedenk bij zo'n werkwijze hoe je je programma's in Subs en Functions kunt opsplitsen. Lees de informatie in de paragrafen 'Werken met Subs' en 'Werken met Functions' eerder in dit hoofdstuk om te bepalen welk type module je gaat gebruiken.

Code schrijven in hapklare brokken is het belangrijkst bij het werken op Sub- en Function-niveau. Gebruik je Subs en Functions als containers voor specifieke taken. Neem bijvoorbeeld legostenen. Elke legosteen is een afzonderlijke eenheid. Schrijf Subs en Functions zodanig dat je ze gemakkelijke kunt splitsen en van elk onderdeel een afzonder-

lijke eenheid kunt maken. De voorbeelden in dit boek demonstreren verschillende manieren om je code in stukken op te breken, aangezien deze vaardigheid bij programmeren het belangrijkst is.

Het is niet moeilijk om erachter te komen hoe je eenvoudige code in stukjes kunt splitsen. De MsgBox-functie die je al vaak hebt gebruikt, is daar een goed voorbeeld van. Ga na hoe deze functie werkt en het berichtvenster op het scherm weergeeft. Je hoeft je er niet om te bekommeren hoe MsgBox deze taak realiseert en er zijn geen eigenaardige voorwaarden om MsgBox te kunnen gebruiken.

Instructies schrijven

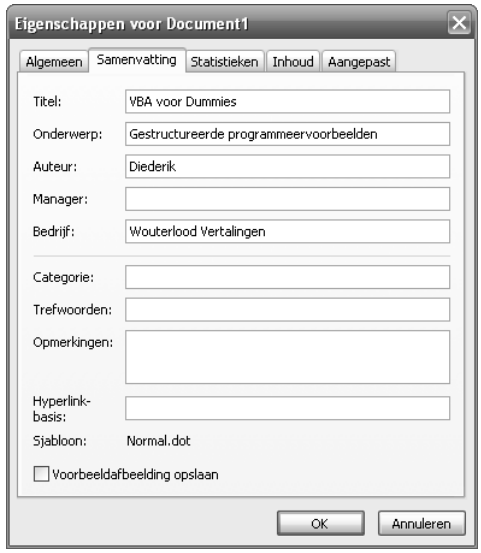
Zodra alle andere ordeningswerkzaamheden achter de rug zijn, heb je een of meer projecten die een of meer modules bevatten, die op hun beurt minstens één Sub bevatten. Al je legostenen heb je in elkaar gezet, maar ze zijn nog leeg. De instructies die je schrijft, vormen uiteindelijk het programma. Alle organisatiewerkzaamheden maken de taak om instructies te schrijven gemakkelijker, omdat je je slechts op één taak tegelijk hoeft te concentreren.

In hoofdstuk 2 benadruk ik het belang om bij het schrijven van instructies een ordelijke werkwijze te volgen. Je begint met een pseudocodeprocedure op te stellen en daarna verander je die pseudocode in instructies die voor VBA begrijpelijk zijn. Dit proces geeft jouw programma twee niveaus van organisatie: code en documentatie. Er zijn echter nog twee andere vormen van organisatie. In de paragraaf 'Leesbare code schrijven' verderop in dit hoofdstuk leg ik uit hoe je commentaar gebruikt om te beschrijven hoe je code werkt.

Je eerste Sub schrijven

De meeste programma's van Microsoft Office bieden een eigenschappenvenster (zie figuur 3.10 voor een voorbeeld) met een tabblad Samenvatting voor documenten. In de meeste programma's van andere fabrikanten vind je een variant op dat eigenschappenvenster. Het tabblad Samenvatting kan een heleboel waardevolle informatie over je programma's bevatten, bijvoorbeeld elementaire gegevens, zoals de naam van de auteur en het bedrijf waar het document werd vervaardigd. Vaak tref je er ook statistische gegevens aan, zoals het aantal woorden in het document. Zie het Help-onderwerp BuiltinDocumentProperties in het VBA-hulpbestand voor aanvullende informatie.

Helaas wordt het dialoogvenster Eigenschappen verborgen door Office 2007-programma's die met het lint werken. Ga in die gevallen als volgt te werk om het dialoogvenster Eigenschappen weer te geven:



Figuur 3.10: Veel toepassingen hebben een tabblad Samenvatting

1. Klik op de Office-knop om het Office-menu te openen.
2. Kies Voorbereiden ➪ Eigenschappen.

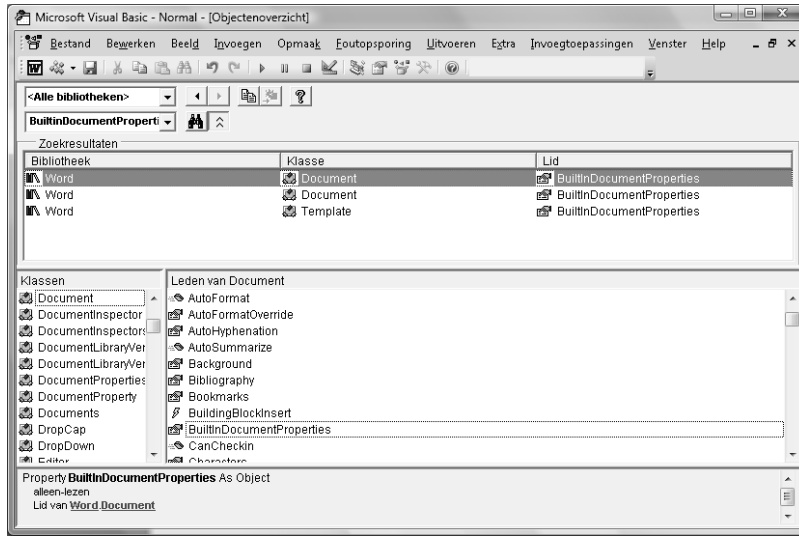
   Het Office-programma toont de lijst van standaardeigenschappen.
3. Klik op Standaard.

   Je ziet in de geopende vervolgkeuzelijst de optie Geavanceerd.
4. Klik op Geavanceerd.

   Het dialoogvenster Eigenschappen (afgebeeld in figuur 3.10) wordt geopend.

Dit is het eerste voorbeeld waarbij je direct met een object werkt. De eigenschap die we zullen gebruiken, heet BuiltinDocumentProperties (ingebouwde documenteigenschappen). In de meeste Office-programma's is deze eigenschap beschikbaar, maar in elk programma is de eigenschap aan een ander object gekoppeld. In Word bijvoorbeeld is deze eigenschap gekoppeld aan de objecten Word.Document en Word.Template, in Excel aan het object Excel.Worksheet. Zoek in de Object Browser (druk zo nodig op F2 om de Object Browser weer te geven), die in hoofdstuk 1 wordt besproken, om deze eigenschap van het Office-programma te vinden. Typ `BuiltinDocumentProperties` in het zoekvak en klik daarna op Zoeken. In figuur 3.11 zie je kenmerkende zoekresultaten voor Excel.

Let op de tekst aan de onderzijde van figuur 3.11. Deze informatie omvat de volledige objectnaam van de eigenschap, wat handig is wanneer je



Figuur 3.11: De documentatie meldt je vaak interessante eigenschappen, maar zegt niet waar je ze kunt vinden

de code schrijft die in figuur 3.11 wordt weergegeven. Dit is de code die je nodig hebt om dit voorbeeld in Excel te maken. Wijzig het object ActiveWorkbook in het object dat door jouw toepassing wordt ondersteund, bijvoorbeeld Document of Template in Word, wanneer je een andere toepassing gebruikt.

Listing 3.2: Auteursinformatie over een document opvragen

```
Public Sub PakSamenvatting()
    'Declareer een DocumentProperty-object dat de informatie zal bevatten.
    Dim MijnEigenschap As DocumentProperty
    'Stel het object DocumentProperty gelijk aan de auteursinformatie.
    Set MijnEigenschap = _
        ActiveWorkbook.BuiltinDocumentProperties("Author")
    'Toon een berichtvenster met de waarde van de eigenschap.
    MsgBox MijnEigenschap.Value, vbOKOnly, "Naam van de auteur"
End Sub
```

Dit voorbeeld begint de declaratie van een variabele met de naam MijnEigenschap. MijnEigenschap verschilt echter van andere variabelen omdat het in feite een object is. Het is een DocumentProperty-object dat een willekeurige documenteigenschap kan bevatten, bijvoorbeeld de auteur of de bedrijfsnaam.

In de volgende regel code wordt MijnEigenschap gelijkgesteld aan de auteursinformatie die door het object BuiltinDocumentProperties ("Author") wordt geleverd. Het is als het ware kijken naar twee appels: je kunt zeggen dat een appel op een andere appel lijkt, maar je kunt niet zeggen dat een appel op een sinaasappel lijkt. In dit geval sla je de auteursinformatie die zich in MijnEigenschap bevindt, op in het object BuiltinDocumentProperties ("Author").

Als een coderegel te lang is, mag je hem in VBA op de volgende regel voortzetten door een underscoreteken (_) toe te voegen (zie listing 3.2 voor een voorbeeld). De underscore is een *voortzettingsteken*. Voeg een voortzettingsteken toe als de tekst op het scherm bladeren vereist. Je kunt de code gemakkelijker lezen als je niet van links naar rechts hoeft te bladeren.

De laatste regel code geeft de eigenschap Value van het object MijnEigenschap weer. VBA weet al hoe er moet worden omgegaan met de andere variabelen. Een object vereist gewoonlijk speciale behandeling. In dit geval vraag je MijnEigenschap wat zijn waarde is om die waarde in een berichtvenster te tonen.

Voer dit programma uit. Je ziet dan een dialoogvenster met de naam van degene die het document maakte. In de meeste gevallen ben jij de maker. Probeer het item Author (zie figuur 3.1) in iets anders te wijzigen. De uitvoer van het programma verandert dan overeenkomstig met wat je in dit veld hebt getypt.

Je eerste functie schrijven

Het voorbeeld uit de vorige paragraaf (zie listing 3.2) is goed, maar misschien wil je wel meer dan een stukje informatie hebben. VBA-gebruikers maken meestal gebruik van functies om herhaalde taken uit te voeren (zie de paragraaf 'Werken met functies' eerder in dit hoofdstuk). Daarom demonstreert het volgende voorbeeld hoe je met functies werkt.

Listing 3.3 gebruikt een Sub, PakSamenvatting2 genoemd, om de functie GetDocProperty meerdere malen op te roepen. Elke keer wordt het resultaat in een speciale variabele opgeslagen. Aan het einde van het programma toont PakSamenvatting2 alle informatie die het programma heeft vergaard.

Listing 3.3: Een functie gebruiken om documentinformatie op te halen

```
Public Sub PakSamenvatting2()
    ' Declareer een string die de uitvoerinformatie bevat.
    Dim DocumentData As String
    ' Sla de naam van de informatie op.
    DocumentData = "Naam van de auteur: "
    ' Haal de naam van de auteur op.
    DocumentData = DocumentData + GetDocProperty("Author")
    ' Voeg een extra regel toe.
    DocumentData = DocumentData + vbCrLf
    ' Sla de naam van de informatie op.
    DocumentData = DocumentData + "Bedrijf: "
    ' Haal de bedrijfsnaam op.
    DocumentData = DocumentData + GetDocProperty("Company")
```

```
    ' Toon een berichtvenster met de eigenschapswaarde.
    MsgBox DocumentData, vbOKOnly,  "Samenvatting"
End Sub
Private Function GetDocProperty(Naam AsString) As String
    ' Declareer een DocumentProperty-object dat de informatie bewaart.
    Dim MyProperty As DocumentProperty
    ' Stel het DocumentProperty-object gelijk aan de auteurinformatie.
    Set MyProperty = ActiveWorkbook.BuiltinDocumentProperties(Name)
    ' Retourneer de informatie.
    GetDocProperty = MyProperty.Value
End Function
```

Listing 3.3 begint met PakSamenvatting2. Een groot deel van wat je hier ziet, lijkt sterk op voorgaande voorbeelden. Let er echter op hoe de code met DocumentData werkt. Het voorbeeld stelt de tekstuitvoer samen door de vorige inhoud van de string aan zichzelf toe te voegen. DocumentData bevat dus om te beginnen "Author Name :", waarna je de feitelijke naam van de auteur eraan toevoegt met behulp van GetDocProperty.

Een andere nieuwe toevoeging is het gebruik van een constante. De constante vbCrLf bevat speciale tekens die hetzelfde doen als wanneer je aan het einde van een tekstregel in een tekstverwerker op de Enter-toets drukt.

De functie GetDocProperty introduceert een paar nieuwe ideeën. Het eerste idee is een retourwaarde: functies kunnen aan de oproeper een retourwaarde overhandigen. Het tweede idee is het gebruik van een argument. Een argument is invoer naar een Sub of een Function. In dit geval is Naam de invoer naar de functie GetDocProperty.

De feitelijke code binnen de functie GetDocProperty verschilt nauwelijks van het voorbeeld in de paragraaf 'Je eerste Sub schrijven' eerder in dit hoofdstuk. In dit geval gebruikt de code in plaats van een constante echter een variabele (Naam) als invoer voor ActiveWorkbook.BuiltinDocumentProperties. Let er ook op dat de code de functie gelijk maakt aan de eigenschap MyProperty.Value. Dit is de manier waarop je een waarde naar de oproeper retourneert.

Bereik binnen bereik

Misschien vind je het begrip *bereik* (*scope*) maar verwarrend en lastig te begrijpen omdat het zo ingewikkeld lijkt. In feite is bereik echter simpelweg wat een programma kan zien en hoe veel aan anderen wordt getoond. Wanneer je de MsgBox-functie bekijkt, houd je je bezig met de invoer die je verstrekt en de uitvoer die de functie produceert. Dit zijn de public (oftewel zichtbare) onderdelen van de MsgBox-functie. Je maalt er niet om wat zich binnen de MsgBox-functie afspeelt, ook al is deze informatie belangrijk voor de MsgBox-functie zelf. Deze innerlijke machinerie,

die je niet kunt zien, betreft de private (oftewel onzichtbare) onderdelen van de MsgBox-functie.

Er zijn twee redenen waarom bereik voor jou als VBA-gebruiker belangrijk is. Ten eerste, als elk gedeelte van een programma elk ander gedeelte van een programma zou kunnen zien, zou dat tot chaos leiden omdat er te veel informatie zou moeten worden bijgehouden. Ten tweede moeten programma's hun gegevens beschermen om er zeker van te zijn dat ze niet op een of andere manier worden beschadigd. Kortom, je wilt sommige gedeelten van je programma zichtbaar maken zodat mensen er gebruik van kunnen maken, maar andere gedeelten onzichtbaar laten zodat ze beschermd blijven.

Het doel van bereik

Je hebt kennisgemaakt met twee sleutelwoorden die tot dusverre voor alle voorbeelden over Sub en Function werden gebruikt: Public en Private. Deze twee sleutelwoorden kunnen ook invloed hebben op andere onderdelen. Je kunt ze gebruiken om het bereik van variabelen of klassen te definiëren. Bereik heeft invloed op bijna elk programmeeronderdeel dat je in dit boek gebruikt, dus is het belangrijk dat je begrijpt hoe bereik werkt.

- **Public** deelt VBA mee dat alle andere programmaonderdelen de desbetreffende onderdelen mogen zien.
- **Private** deelt VBA mee dat de desbetreffende onderdelen voor andere programmaonderdelen moeten worden verborgen.

De effecten van bereik bepalen

Met bereik werken is de beste manier om ermee vertrouwd te raken. Je kunt met eenvoudige gevallen van bereik experimenteren om na te gaan hoe een verandering van bereik op jouw programma's van invloed is. De belangrijkste regel is dat bereik uitsluitend van invloed is op alles buiten het huidige blok. Dat is juist, het legovoorbeeld werkt ook hier. (Zie de paragraaf 'Leve de legoaanpak' eerder in dit hoofdstuk.)

Als je een module private maakt door Option Private Module aan het begin van de module toe te voegen, houdt dat in dat alles in die module voor de buitenwereld onzichtbaar is. Zelfs als de module een Public Sub bevat, kunnen uitsluitend de andere onderdelen binnen de module die Sub zien; voor alles buiten de module is de Public Sub onzichtbaar. Evenzo zal alles binnen de huidige module een Private Sub kunnen zien als je er eentje declareert, maar is die Sub voor alles buiten de huidige module onzichtbaar.

Listing 3.4 demonstreert enkele basiskenmerken van bereik. In andere voorbeelden van dit boek zal dit concept nog worden uitgediept, maar dit biedt een goed vertrekpunt.

Listing 3.4: Werken met globale variabelen

```
' Declareer een private globale variabele.
Private MijnGlobaleVariabele As String

Public Sub GlobaleTest()
    ' Stel de waarde van de globale variabele in.
    MijnGlobaleVariabele = "Hallo"
    ' Toon de waarde.
    MsgBox MijnGlobaleVariabele
    ' Roep de Sub GlobaleTest2 aan.
    GlobaleTest2
    ' Toon de waarde bij terugkeer van de oproep.
    MsgBox MijnGlobaleVariabele
End Sub

Private Sub GlobaleTest2()
    ' Laat zien dat de globale variabele echt globaal is.
    MsgBox MijnGlobaleVariabele
    ' Wijzig de waarde van de globale variabele.
    MijnGlobaleVariabele = "Tot ziens"
End Sub
```

Bedenk dat MijnGlobaleVariabele private is. Je kunt deze globale variabele niet van buiten de huidige module bereiken. Beide subprocedures binnen deze module kunnen de globale variabele echter wel bereiken.

Je ziet dat GlobalTest een public Sub is, terwijl GlobalTest2 private is. Je kunt het gebruik van bereik in dit geval verifiëren door het dialoogvenster Macro te openen (kies Extra ⇨ Macro ⇨ Macro's). Je ziet dan dat GlobalTest in de lijst staat, maar GlobalTest2 niet.

Typ de voorbeeldcode van listing 3.4 over en voer hem uit om te zien hoe de twee Sub-onderdelen elkaar beïnvloeden. Je dient drie dialoogvensters te zien. Het eerste dialoogvenster zegt Hallo, omdat GlobalTest de waarde van MijnGlobaleVariabele instelt. Het tweede dialoogvenster zegt eveneens Hallo, aangezien MijnGlobaleVariabele echt globaal is. Ook al werd de waarde van deze variabele in GlobalTest ingesteld, toch kan GlobalTest2 deze waarde lezen. Het derde dialoogvenster zegt Tot ziens, omdat GlobalTest2 MijnGlobaleVariabele op een andere waarde instelde.

Leesbare code maken

Misschien is het je opgevallen dat de voorbeelden in dit hoofdstuk van spaties en lege regels gebruikmaken om de code leesbaarder te maken. Als je alle instructies van je programma achter elkaar typt, werkt de code nog steeds. VBA negeert spaties en lege regels simpelweg. Maar jij kunt de code beter de ruimte geven, want code zonder witruimte is praktisch onleesbaar.

De meeste VBA-gebruikers gebruiken twee soorten witruimte. Met het eerste type heb je in dit hoofdstuk kennisgemaakt. Elk commentaar- en instructiepaar in de code wordt gevolgd door een lege regel. Deze lege regel deelt mee aan iedereen die de code leest dat het einde van deze specifieke instructie of stap in de procedure is bereikt.

Het tweede soort witruimte is inspringing. De voorbeelden in dit hoofdstuk laten de code binnen een Sub of Function inspringen om duidelijk aan te geven wat tot de inhoud van de Sub of Function behoort.

Je code verduidelijken voor anderen

De pseudocodemethode die ik in hoofdstuk 2 beschrijf, is een goede manier om met het documenteren van je code te beginnen. Op een bepaald punt zul je echter informatie moeten toevoegen omdat het niet altijd voldoende is om de procedure te lezen die door de pseudocode wordt beschreven. Je wil bijvoorbeeld je naam, een projecttitel en andere vormen van documentatie aan de code toevoegen.

Elementaire commentaarregels schrijven

Commentaar kan er op diverse manieren uitzien. Pseudocode is commentaar dat iedereen schrijft omdat dit de natuurlijkste vorm van toelichting is. Ontwikkelaars gaan dan door met het toevoegen van documentatiecommentaar, bijvoorbeeld wie het programma heeft geschreven of wanneer het oorspronkelijk werd geschreven, en een lijst van updates die aan de code zijn aangebracht. Sommige ontwikkelaars gaan vervolgens nog een stap verder en schrijven uitgebreider commentaar.

Een van de belangrijkste commentaarregels die je aan je code kunt toevoegen is waarom je ervoor koos om het programma op een bepaalde manier te schrijven. Simpelweg zeggen dat de code een specifieke taak uitvoert is niet voldoende, aangezien je een en dezelfde taak gewoonlijk op verschillende manieren kunt uitvoeren. Aangeven waarom je bepaalde keuzen hebt gemaakt, kan het aantal fouten tijdens updates verminderen of juist als reden dienen om naderhand updates aan te brengen, op een moment dat je codeertechniek beter is geworden.

Als goede programmeur voeg je ook gemaakte fouten als commentaar aan de code toe, met name als je denkt dat iemand anders dezelfde fout kan maken. Aan dit ervaringscommentaar heb ik in veel situaties veel gehad, omdat ik ze uiteindelijk als notities ben gaan gebruiken. Wanneer ik met een nieuw project begin, neem ik mijn aantekeningen door, op zoek naar dingen die ik moet vermijden.

Weten wanneer je commentaar gebruikt

Gebruik commentaar wanneer en waar je dat nodig vindt. Misschien denk je dat commentaarregels lastig te typen zijn en dat je bij elk programma slechts een of twee keer beknopte commentaarregels moet bijvoegen. Je hebt gelijk, goed commentaar schrijven kan veel tijd in beslag nemen en kan bovendien moeilijk zijn, aangezien je tijdens het schrijven weer over de code na moet denken. Maar de programma's met het minste commentaar produceren tijdens een update gewoonlijk de meeste hoofdbrekens. Ik heb het in de praktijk zelfs een paar keer meegemaakt dat de afwezigheid van commentaar bij de code tot gevolg had dat een bedrijf de code van een update weer van het begin af aan liet schrijven in plaats van iemand ervoor te betalen om moeizaam uit te vissen wat de oude code betekende.

Begrijpen hoe je een goed commentaar opstelt

Een goed commentaar is een toelichting die je begrijpt. Gebruik geen moeilijke woorden; schrijf alles op in gewone termen die je kunt begrijpen. Als je het gevoel hebt dat je iets moet uitleggen, doe dat dan gerust. Goed commentaar dient een antwoord te geven op de zes essentiële vragen: wie, wat, waar, wanneer, waarom en hoe. Zorg ervoor dat je opmerkingen volledig zijn en dat ze een volledig antwoord geven op eventuele vragen die zouden kunnen rijzen bij iemand die jouw code leest.