Basic Bulletin

23ste jaargang maart 2016 Nummer 1

HCC BASIC interessegroep

Inhoud

Contacten

Functie	Naam	Telefoonnr.	E-mail
Voorzitter	Jan van der Linden	071-3413679	voorz@basic-gg.hcc.nl
Secretaris	Gordon Rahman Tobias Asserstraat 6 2037 JA Haarlem	023-5334881	secr@basic-gg.hcc.nl
Penningmeester	Piet Boere	0348-473115	penm@basic-gg.hcc.nl
Bestuurslid	Titus Krijgsman	075-6145458	t.krijgsman8@upcmail.nl
Redacteur	M.A. Kurvers Schaapsveld 46 3773 ZJ Barneveld	06-30896598	m.a.kurvers@live.nl
Ledenadministratie	Fred Luchsinger	0318-571187	f.luchsinger@kader.hcc.nl
Webmaster	Jan van der Linden	071-3413679	j.vd.linden@kader.hcc.nl

http://www.basic.hcc.nl

Redactioneel

In dit nieuwe jaar wil ik weer leuke voorbeelden laten zien wat we allemaal kunnen doen met Power-BASIC en Excel.

PowerBASIC is het enige BASIC dialect die zeer veel, en krachtige, variabelentypes heeft. Het gebruik van dynamische recordvelden is voor BASIC daarom uniek.

Voor Excel is het gebruik van VBA soms onnodig. Een professioneel die veel van Excel weet, kan de toepassing helemaal ombouwen met gebruik van VBA. Gelukkig kan de code niet gecompileerd worden naar een toepassing. Excel zelf wordt ook vaak vergeten, daarom wil ik laten zien dat ook op het werkblad veel functies bestaan, zelfs functies die in VBA niet aanwezig zijn. Dit betekent niet dat dan ook alles op het werkblad beschikbaar is naar wat wij willen doen.

Marco Kurvers

BBC BASIC for Windows – De library's (8).

Kalender functies

De **DATELIB** bibliotheek bevat een set van functies voor het gebruik van de datum operaties. De bibliotheek moet worden geladen vanuit uw programma met behulp van de opdracht:

```
INSTALL @lib$+"DATELIB"
```

De functies zijn:

- FN_mjd
- FN_day
- FN_month
- FN_year
- FN_dow
- FN_dim
- FN_today
- FN_date$
- FN_readdate

FN_mjd(day%, month%, year%)

Deze functie neemt een datum (bestaande uit een dag-van-de-maand, een maand en een jaar) en converteert deze naar het corresponderende Modified Julian Day nummer. De Modified Julian Day begon met tellen om middernacht (00:00 uur) op woensdag 17 november 1858 (waardoor MJD is 0). Dagen vóór die datum hebben negatieve MJD getallen. U kunt gemakkelijk het aantal dagen tussen twee verschillende datums berekenen door de Modified Julian Day getallen af te trekken.

De opgegeven parameters zijn de dag van de maand (1-31), het maand nummer (1-12) en het jaar nummer (1-9999). Merk op dat de functies in de DATELIB bibliotheek consequent voor een datum in dat bereik zich gedragen zal (bijvoorbeeld, converteren van DMJ naar MJD en terug zullen de oorspronkelijke waarden terugkeren), maar normaal gesproken mag het niet worden gebruikt voor datums vóór de invoering van de Gregoriaanse kalender (in het Verenigd Koninkrijk op donderdag 14 september 1752, MJD –38779). Voor eerdere datums kunnen de dag, maand en jaar waarden niet juist zijn en sinds het gebruik van de oude Julian kalender in sommige landen die zo laat pas in 1927 werd vastgelegd, moet zorg worden genomen bij het gebruik van deze functie.

FN_day(mjd%)

Deze functie neemt een Modified Julian Day nummer en geeft de corresponderende dag-van-de-maand (1-31) terug.

FN_month(mjd%)

Deze functie neemt een Modified Julian Day nummer en geeft de corresponderende maand (1-12) terug.

FN_year(mjd%)

Deze functie neemt een Modified Julian Day nummer en geeft het corresponderende jaar (1-9999) terug.

FN_dow(mjd%)

Deze functie neemt een Modified Julian Day nummer en geeft de corresponderende dag-van-de-week (0-6, waarvan 0 is zondag) terug.

FN_dim(month%, year%)
Deze functie neemt een maand (1-12) en een jaar (1-9999) en geeft het aantal dagen in de maand (in het bereik 28 tot 31). Bij het instellen van **month%** naar 2 (februari) kunt u deze functie gebruiken om te bepalen of het jaar een schrikkeljaar is.

FN_today
Deze functie geeft het corresponderende Modified Julian Day van de datum van vandaag (ervan uitgaande dat de PC klok correct is ingesteld).

FN_date$(mjd%, format$)
Deze functie neemt een Modified Julian Day nummer en een ingedeelde tekenreeks en geeft een ingedeelde tekenreeks terug met als inhoud de datum. De ingedeelde tekenreeks kan de volgende codes bevatten:

d	Dag van de maand als cijfers zonder voorloop nul.
dd	Dag van de maand als cijfers met voorloop nul voor dagen met één cijfer.
ddd	Dag van de week als een drieletter afkorting.
dddd	Dag van de week als de volle naam.
M	Maand als cijfers zonder voorloop nul.
MM	Maand als cijfers met voorloop nul voor maanden met één cijfer.
MMM	Maand als een drieletter afkorting.
MMMM	Maand als de volle naam.
y	Jaar als laatste twee cijfers, maar zonder voorloop nul.
yy	Jaar als laatste twee cijfers, maar met voorloop nul voor jaren kleiner dan 10.
yyyy	Jaar vertegenwoordigd bij vier volledige cijfers.

Als voorbeeld:

```
date$ = FN_date$(mjd%, "ddd dd MMM yyyy")
```

geeft een tekenreeks terug in de vorm “Sun 22 Feb 2004”.

FN_readdate(date$, code$, minyear%)
Deze functie verdeelt een tekenreeks die een datum bevat en geeft het corresponderende Modified Julian Day nummer terug. Naast de datum moet u een tekenreeks met een van de volgende codes opgeven: "DMJ", "MDJ", "JMD", "JDM", "DJM" of "MJD". Dit informeert de functie van de volgorde waarin de verschillende elementen van de datum (dag, maand, jaar) in de tekenreeks aanwezig zijn. Als het jaar opgegeven is met een viercijferig nummer, zal de derde parameter niet worden gebruikt. Als alleen de laatste twee cijfers van het jaar opgegeven zijn, is de derde parameter het *minimum* jaar nummer die teruggegeven wordt. Bijvoorbeeld, als de derde parameter is 1950, zullen twee cijfers van het jaar nummer 1950 corresponderen tot inclusief 2049.

De **FN_readdate** functie probeert de indeling van de tekenreeks op te maken, zolang de elementen in de opgegeven volgorde zijn. Bijvoorbeeld, het accepteert “22/2/2004”, “22 Feb 04”, “22-02-04” enz. Als de tekenreeks niet opgemaakt kan worden, wordt er een &80000000 waarde teruggegeven.

Direct3D graphics

De **D3DLIB** bibliotheek bevat een set van procedures en functies voor het weergeven en animeren van driedimensionale afbeeldingen. Het biedt een interface voor Microsofts **Direct3D**[tm] en vereist *DirectX* versie 8.0 of hoger geïnstalleerd. De bibliotheek moet worden geladen vanuit uw programma met gebruik van het commando:

```
INSTALL @lib$+"D3DLIB"
```

De procedures en functies zijn:

- FN_initd3d
- FN_load3d
- FN_loadtexture
- PROC_release
- FN_f4
- PROC_render

FN_initd3d(hw%, cull%, light%)

Deze functie zal het Direct3D systeem initialiseren en geeft een pointer terug naar een Direct3D device. Als de teruggegeven waarde nul is, dan zal aangeven worden dat DirectX 8.0 of hoger niet geïnstalleerd is.

De waarde **hw%** is de *handle* van het venster die de 3D afbeeldingen zal bevatten. Het kan worden ingeschakeld naar **@hwnd%** als de afbeeldingen worden weergegeven in het hoofd uitvoervenster van BBC BASIC of naar de handle van een *zoonvenster* (bijvoorbeeld als teruggeven vanuit FN_staticbox) als u ze wilt laten verschijnen in een afzonderlijk kader. U kunt echter niet Direct3D afbeeldingen en normale BBC BASIC uitvoer (tekst of afbeeldingen) in hetzelfde venster mixen.

De waarde **cull%** geeft de *culling mode* aan, waarmee bepaald wordt of oppervlakken zich als *enkelzijdig* of *dubbelzijdig* gedragen. Mogelijke waarden zijn 1 (geen), 2 (met de klok mee) of 3 (tegen de klok in). In geval van twijfel, stel het dan in op 1.

De waarde **light%** bepaald of de verlichting motor van Direct3D is ingeschakeld. Schakel het in met 1 voor verlichting aan of 0 voor verlichting uit. Wanneer de verlichting uitgeschakeld is, zullen alle objecten normaal worden weergegeven met een gelijkmatige verlichting. Wanneer de verlichting ingeschakeld is, dan is het nodig voor alle objecten de *normale oppervlakken,* die in de vertex descriptie staat, op te nemen.

FN_load3d(pdev%, file$, num%, fmt%, size%)

Deze functie laadt een object of scene in (bestaande uit een aantal triangels, elk bestaande uit drie vertices) vanuit een bestand. Vertices zijn hoekpunten in het Nederlands vertaald, maar de naam vertices wordt in de computertechniek meer gebruikt. Een hoekpunt is een vertex. De functie retourneert een pointer naar een vertex buffer. Als de functie een nul retourneert, dan kan het bestand niet worden geopend.

De waarde **pdev%** is de pointer die verkregen is van FN_initd3d. De waarde **file$** is de naam van een bestand die de vertex gegevens bevat.

De waarden **num%**, **fmt%** en **size%** zijn de *uitvoeringen* vanuit FN_load3d en worden bepaald voor het aantal vertices, het vertex formaat en de grootte in bytes van elke respectievelijke vertex.

Het bestandsformaat is als volgt:

```
Aantal vertices (4 bytes, LSB eerst)
Vertex formaat (2 bytes, LSB eerst)
Vertex size in bytes (2 bytes, LSB eerst)
Gegevens voor elke vertex (zie hieronder)
```

Vertex descriptie
De vertex gegevens moeten voldoen aan de eisen van het Direct3Ds *flexibele vertex formaat*. Elke vertex bestaat uit een of meer van de volgende items, in de aangegeven volgorde:

Code	Grootte	Gegevens	Omschrijving
&002	12	XYZ positie	Altijd vereist
&010	12	Normale oppervlakte	Wanneer verlichting gebruikt
&040	4	Diffuse kleur	Wanneer textuur noch materiaal opgegeven
&080	4	Spiegelende kleur	Voor glanzende voorwerpen
&100	8	UV textuur coördinaten	Wanneer textuur opgegeven

Om de vertex formaat code te verkrijgen, voeg de codes van de items samen die ingevoegd zijn. Om de vertex grootte te verkrijgen, voeg de grootte van de items toe die ingevoegd zijn. De **XYZ positie** en **normale oppervlakte** items bestaan elk uit drie 4-byte floating point nummers (zie FN_f4). De **diffuse kleur** en **spiegelende kleur** items bestaan elk uit 4-byte kleurwaarden (&FFrrggbb). De **textuur coördinaten** bestaan uit een paar van 4-byte floating point nummers. De makkelijkste vertex descriptie bestaat uit een **XYZ positie** en een **diffuse kleur** (formaat &042; grootte 16 bytes). Zie FN_f4 voor een voorbeeld hoe een bestand in dit formaat wordt gemaakt. U kunt ook kijken op Microsoft documentatie voor meer details.

FN_loadtexture(pdev%, file$)
Deze functie laadt een textuur map in vanuit een afbeeldingsbestand (BMP, ICO, GIF, JPEG, EMF of WMF formaat) en retourneert een pointer naar de textuur. Als de waarde nul teruggegeven is, kon het bestand niet worden geopend of worden erkend. De bestandsnaam moet beginnen met een stationsletter om het met een web URL te onderscheiden.

De waarde **pdev%** is de pointer die verkregen is vanuit FN_initd3d. De waarde **file$** is de naam van het afbeeldingsbestand.

De afbeelding zal worden opgevuld tot een grootte van 2 ^ n pixels in zowel horizontale en verticale richtingen.

PROC_release(pobj%)
Deze functie geeft een object vrij (Direct3D device, vertex buffer of textuur) wanneer het niet langer vereist is. Het wordt gebruikt om de resources gebruikt bij de objecten vrij te maken, anders kunt u uiteindelijk uit het geheugen komen.

De waarde **pobj%** is de pointer teruggegeven vanuit FN_initd3d, FN_load3d of FN_loadtexture.

FN_f4(num)
Deze functie converteert een nummer naar een 4-byte (enkel precisie) floating point waarde. Alle floating point waarden gebruikt bij Direct3D (bijvoorbeeld binnen de vertex descripties) zijn in dit formaat. Als een illustratie van dit, gebruik het volgende code segment dat een bestand maakt met als inhoud een enkele triangel bestaande uit drie vertices, geschikt voor het laden bij FN_load3d:

```
F% = OPENOUT"TRIANGLE.B3D"
PROC4(3):REM 3 vertices
PROC4(&100042):REM vertex grootte &10 en formaat &42
```

```
PROC4(FN_f4(-1.0))
PROC4(FN_f4(-1.0))
PROC4(FN_f4(1.0))
PROC4(&FF0000FF)
PROC4(FN_f4(1.0))
PROC4(FN_f4(-1.0))
PROC4(FN_f4(1.0))
PROC4(&FF00FF00)
PROC4(FN_f4(0.0))
PROC4(FN_f4(1.0))
PROC4(FN_f4(0.0))
PROC4(&FFFF0000)
CLOSE #F%
DEF PROC4(A%)
     BPUT#F%,A%
     BPUT#F%,A%>>8
     BPUT#F%,A%>>16
     BPUT#F%,A%>>24
ENDPROC
```

PROC_render(pdev%, ….)

Deze procedure tekent een 2D beeld van de 3D wereld op het scherm. Het neemt **23** parameters als volgt:

pdev%	De waarde teruggegeven vanuit FN_initd3d.
bcol%	De achtergrondkleur (&&FFrrggbb).
nlight%	Het aantal verlichtingen. Stel in als nul als verlichting niet nodig is.
light%()	Een array van pointers naar D3DLIGHT8 structuren (zie notitie 1).
nobj%	Het aantal objecten (dat wil zeggen: vertex buffers).
mat%()	Een array van pointers naar D3DMATERIAL8 structuren (zie notitie 2).
tex%()	Een array van textuur pointers (bijvoorbeeld, teruggegeven vanuit FN_loadtexture).
vbuf%()	Een array van vertex buffer pointers (bijvoorbeeld, teruggegeven vanuit FN_load3d).
vnum%()	Een array van totaal aantal vertices (bijvoorbeeld, teruggegeven vanuit FN_load3d).
vfmt%()	Een array van vertex formaat codes (bijvoorbeeld, teruggegeven vanuit FN_load3d).
vsize%()	Een array van vertex grootte (bijvoorbeeld, teruggegeven vanuit FN_load3d).
yaw()	Een array van *yaw* hoeken (rotaties over de Y-as).
pitch()	Een array van *pitch* hoeken (rotaties over de X-as).
roll()	Een array van *roll* hoeken (rotaties over de Z-as).
X()	Een array van translaties langs de X-as.
Y()	Een array van translaties langs de Y-as.
Z()	Een array van translaties langs de Z-as.
eye()	Een array eye(0), eye(1), eye(2) houdt de XYZ coördinaten vast van het oog of camera.
look()	Een array look(0), look(1), look(2) houdt de XYZ coördinaten vast van een punt op de ooglijn.
fov	Het verticale beeldveld in radialen (gelijk aan de camera's zoom).
ar	De aspect ratio van de 3D afbeeldingsvenster (breedte/hoogte).
zn	De afstand vanuit de camera tot de dichtstbijzijnde ruimte (objecten dichterbij dan dit zijn onzichtbaar).
zf	De afstand vanuit de camera tot de verste ruimte (objecten verder weg dan dit zijn onzichtbaar).

Notities:

1. Een Direct3D directionele licht kan als volgt worden gemaakt. Raadpleeg ook de Microsoft documentatie voor ander licht types.

```
DIM light%(0) 103
light%(0)!0 = 3 : REM directionele licht
light%(0)!4 = FN_f4(1)     : REM rode component
light%(0)!8 = FN_f4(1)     : REM groene component
light%(0)!12 = FN_f4(0)    : REM blauwe component
light%(0)!64 = FN_f4(0)    : REM. X component van afstand
light%(0)!68 = FN_f4(0)    : REM. Y component van afstand
light%(0)!72 = FN_f4(1)    : REM. Z component van afstand
```

2. Een Direct3D materiaal kan als volgt worden gemaakt. Raadpleeg ook de Microsoft documentatie voor ander materiaal types.

```
DIM mat%(0) 67
mat%(0)!0 = FN_f4(1)  : REM rode kleur component
mat%(0)!4 = FN_f4(1)  : REM groene kleur component
mat%(0)!8 = FN_f4(1)  : REM blauwe kleur component
```

3. De arrays van object parameters moeten minstens zo veel elementen hebben als de waarde van **nobj%**. Ongebruikte arrays, als voorbeeld **mat%()** of **tex%()**, moeten nullen bevatten (dit is de oorspronkelijke toestand na DIM).

4. Rotaties nemen plaats om de wereld assen in de volgorde **roll** dan **pitch** dan **yaw**.

Schuine ellipsen plotten

De *BBC BASIC for Windows* ELLIPSE statement plot alleen as-gebonden ellipsen. De **ELLIPSE** bibliotheek bevat de procedures **PROCellipse** en **PROCellipsefill** waarmee een uitlijn of gevulde ellips geroteerd bij een opgegeven hoek geplot kan worden. De bibliotheek zal vanaf uw programma geladen moeten worden met onderstaand commando:

```
INSTALL @lib$+"ELLIPSE"
```

Als alternatief, aangezien de procedures heel kort zijn, wilt u ze misschien in uw eigen programma nemen (gebruik het Insert commando vanuit het File menu).

PROCellipse(x, y, a, b, angle)

Deze procedure plot een uitlijn ellips gecentreerd met de grafische coördinaten **x,y** en met de straallengten **a** en **b**. De vijfde parameter bepaald dat de ellips tegen de klok in geroteerd moet worden bij **angle** radialen (als de hoek nul is, zal de ellips met de **a** as horizontaal geplot worden, net zoals bij het normale ELLIPSE statement).

De ellips is getekend in de huidige grafische voorgrondkleur en mode, zoals opgegeven bij GCOL.

PROCellipsefill(x, y, a, b, angle)

Deze procedure werkt op dezelfde manier als PROCellipse, maar nu zal de ellips gevuld worden geplot. Nogmaals, de ellips is getekend in de huidige grafische voorgrondkleur en mode, zoals opgegeven bij GCOL.

In de volgende Bulletin komen de sleutelwoorden aan bod. U kunt dan de structuur van BBC BASIC for Windows verkennen.

XNA Game Studio 4.0 – Sprite objecten en Namespaces.

De grote bibliotheken .NET Framework en XNA Game Studio Framework geven structuur aan de code in Visual Studio. De manier hoe de bibliotheken opgebouwd zijn laat zien hoe we het uit kunnen breiden met onze eigen code.
Projecten worden gemaakt om deze uitbreidingen in andere projecten in te kunnen voegen. Eerder konden we nog van projecten spreken als er een programma geschreven werd, maar tegenwoordig is een project niet meer het hele programma waar alles om draait.

Sprite objecten

In het VBGame programma was er een sprite te zien die stuiterend op het scherm bewoog. De sprite is alleen maar een afbeelding. Het bevat geen gegevens, eigenschappen, gebeurtenissen, niets. Toch kunnen we de sprite besturen op de manier zoals we willen, maar stel dat we meer soorten sprites willen laten bewegen. We zouden dan een heleboel **Texture2D** variabelen moeten aanmaken en ook meer **Vector2** variabelen. De Update() methode zou een gigantische codeblok worden met allemaal besturingscontrole. In het Nederlands vertaald kan ik zeggen dat we dit kunnen vergelijken met 10 soorten fruit waarvan elk fruit zijn eigen fruitmand heeft.

In principe werkt de techniek ook bij het maken van gewone Windows applicaties. Een goede OOP betekent een code dat gemakkelijk te volgen is. Er is dus geen verschil tussen deze twee bibliotheken, alleen de besturing om de code te laten werken is anders.

In Visual C# staan de sprite klassen (en andere soort klassen) in aparte projecten. Dit kan ook in Visual Basic worden gedaan. Voordat we gaan kijken hoe we dat doen, is er eerst iets wat we moeten wijzigen.

Namespaces

Elke klasse wordt door Visual C# in een namespace geplaatst met de opgegeven projectnaam. Elke klasse die toegevoegd wordt komt in dezelfde namespace, zolang het project dezelfde is. Zie hieronder een voorbeeld van een solution die uit meerdere projecten bestaat.

Solution A: `opstartproject A`

Project A	**Project B**	**Project C**
Program klasse (of module: VB)	Klasse Base (niet persé)	Klasse Base (niet persé)
Klasse A (hoofdklasse)	Klasse A	Klasse A
In klasse A: Klasse B, Project B en Project C invoegen. Klasse B	Klasse B Klasse C Klasse A, B en C e.v.t. zoonklassen van klasse Base.	Klasse B Klasse C Klasse D Klasse E
		Klassen A t/m E e.v.t. zoonklassen van klasse Base.

Net zoals bij het automatisch aanmaken van de sjabloon (dat in Visual Basic niet gebeurt), is het bij de namespaces hetzelfde probleem. Zodra we in Visual Basic een nieuw project toe willen voegen en een klassenbibliotheek (class library) kiezen, wordt er een project aan de solution toegevoegd met de opgegeven projectnaam en een klasse, genaamd *Class1* echter zonder de namespace. Als we de twee XNA sjablonen van Visual C# en Visual Basic nog eens bekijken, zien we daar ook het verschil: het ontbreken van de namespaces in Visual Basic.

Maar er is nog een reden waarom we toch namespaces moeten gebruiken, ook al werkt de sjabloon in Visual Basic goed. De reden is: het communiceren met andere klassen die in andere projecten staan.

Zonder namespaces kunnen we de klassen in Project B en C via Project A niet bereiken. Onderstaand tabel laat nog een solution zien van een compleet BreakOut game.

`Solution BreakOut: opstartproject BreakOut`

BreakOut project	**SpriteLib** projectbibliotheek	**StuffLib** projectbibliotheek
Program klasse (of module: VB)	Spriteklasse **Base** (niet persé)	Klasse **Base** (niet persé)
Klasse **Game1** In deze klasse: SpriteLib en StuffLib invoegen.	Spriteklasse **Bal** Spriteklasse **Paddle** Spriteklasse **Brick**	Klasse **Level** Klasse **Sound** Klassen voor extra gegevens.
	Eventueel kunnen de klassen weer zoonklassen zijn van klasse **Base**.	

Willen we een Windows applicatie of een XNA game programmeren op de manier zoals het tabel aangeeft, dan moeten we zelf namespaces in de projecten aanmaken.

Open het XNA sjabloon in Visual Basic. Als u die nog niet heeft, volg dan de stappen in Bulletin nummer 3 van 2015 om er een te maken.

Knip de hele klasse Game1 uit, exclusief de Imports regels en typ onderstaande code onder de laatste Imports regel:

```
Namespace VBGame

End Namespace
```

Plak de Game1 klasse ertussenin zodat het precies overeenkomt als in Visual C# code staat.

Nu zouden we graag ook een namespace willen hebben in de Program module, want de Program klasse in Visual C# heeft er ook een. Gelukkig kan ook in een module een namespace worden gemaakt.

Ga naar de mdlMain module of Program module en volg dezelfde stappen. Ook in de module moet de naam van de namespace **VBGame** zijn.

! Merk op dat de naam van de solution dezelfde naam is als de naam van het opstartproject. Alle namespaces in de klassen en modules die in het opstartproject staan moeten ook die naam hebben. Dat geldt ook voor de namespaces in de klassen van de andere projecten. Bijvoorbeeld, de **Bal** klasse heeft de namespace **SpriteLib**.

Als u wel VBGame gevolgd heeft in Bulletin nummer 3 en 4 van 2015, dan kunt u bovenstaande stappen ook doen, maar zorg er ook voor dat u de namespaces dus ook in de sjabloon hebt.

Spriteklassen maken

Voordat we objecten hebben, moeten we eerst klassen maken voor de sprites. Het enige wat we moeten doen is alle variabelen, die voor een bepaalde sprite te maken hebben, uit de Game1 te halen en in de juiste nieuwe klasse te plaatsen. Een goed voorbeeld is de bal die een Texture2D variabele en een Vector2 variabele heeft.

Bepaal zelf wat u wilt, de nieuwe klasse toevoegen in hetzelfde opstartproject, waar de klasse Game1 ook in staat, of een nieuw project toevoegen aan de solution en daar de nieuwe klasse in maken. Raadpleeg bovenstaand tabel als u even niet meer goed weet hoe de opzet van de projecten in elkaar zit.

Vergeet niet, mocht u een nieuw project toevoegen, een Class Library als project te kiezen en niet nog eens een Windows Forms Application!

Als het gelukt is, zal de klasse er staan als hieronder:

```
Public Class Class1

End Class
```

Geef de klasse de naam **Bal**.

U kunt eventueel de klasse in een namespace zetten, maar het is niet nodig als de klasse direct vanuit Game1 bereikbaar moet zijn. Anders moet de namespace geïmporteerd worden bij de andere Imports regels in Game1.

Knip onderstaande regels uit **Game1** en plak deze in de klasse Bal.

```
'Texture die we willen renderen
Private texture As Texture2D
'Bepaal waar de sprite getekend wordt
Private spritePos As Vector2 = Vector2.Zero
'X en Y snelheid van de sprite
Private XSpeed As Single = 80
Private YSpeed As Single = 120
'Vector2 wordt gebruikt voor de snelheid van de sprite
Private spriteSpeed As New Vector2(XSpeed, YSpeed)
```

De types Texture2D en Vector2 worden nu niet meer herkend. Ook in deze klasse moeten de juiste namespaces van XNA geïmporteerd worden. Onderstaande twee regels is voor de klasse voldoende. Plaats ze boven de klasse.

```
Imports Microsoft.Xna.Framework
Imports Microsoft.Xna.Framework.Graphics
```

Maar wat u ook probeert, het importeren werkt niet. Elke keer als u een nieuwe klasse aanmaakt, moeten de XNA namespaces opnieuw uit de resources geladen worden. U moet ze dus via het menu Project -> Add Reference… toevoegen, net zoals de eerste keer gedaan is in Bulletin nummer 3 van 2015.

Nu zullen de types wel worden herkend.

Het laden van de sprite zal blijven staan in de LoadContent() methode, maar daar bestaat de variabele **texture** niet meer. Om de besturing van de sprite te laten gebeuren in de klasse, moeten we de sprite aan de klasse *doorgeven*. Oplossing: het maken van een constructor.

Neem onderstaande code over en typ het in de Bal klasse onderaan de variabelen.

```
Public Sub New(ByVal texture As Texture2D)
     Me.texture = texture
End Sub
```

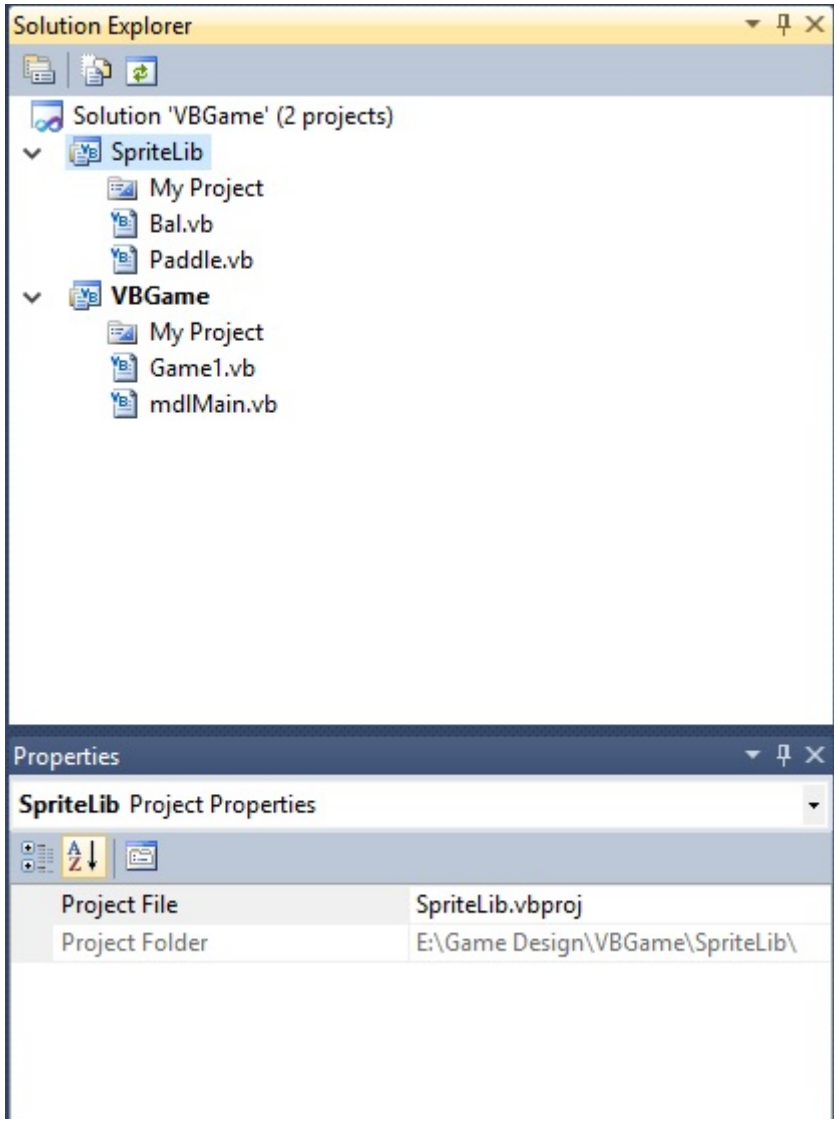
Nu kunnen we een object maken van de sprite. Waar we eerder de variabelen in hadden staan, plaatsen we nu onderstaande regel in Game1:

```
Dim bal As Bal
```

Maar het Bal type wordt niet herkend, mocht u in de klasse een namespace hebben geplaatst. Onderstaande Imports regel is nodig om het type te herkennen:

```
Imports SpriteLib.SpriteLib   'Of een andere gegeven naam
```

Maar ook de Imports regel zal niet werken, totdat we deze uit de *references* hebben geladen. Projecten die u zelf maakt komen terecht in de referenceslijst. Elke keer als u de klassen wilt gebruiken, moet u ze inladen voordat u ze kunt importeren in de code.



! Mocht uw klasse niet in een apart project staan, dan wordt het Bal type wel herkend.

Als u met importeren uw project typt, is het echter niet voldoende. Visual Basic vraagt om de naam nog een keer op te geven, gescheiden met een punt. De twee namen kunnen verschillend zijn. De reden waarom er twee namen opgegeven moeten worden, komt doordat Visual Basic moet weten waar de namespace van het project staat. Dat is de mapnaam die ook SpriteLib heet. Klik maar eens op het project zelf. Bij de eigenschappen ziet u de pad van waar de bestandsnamen van de klassen in staan, zie afbeelding.

Voordat we de besturing in de Bal klasse kunnen plaatsen, moeten we wat wijzigingen aanbrengen in de LoadContent() methode. We gebruiken een lokale **texture** variabele om de sprite in te laden en geven deze aan de constructor van de Bal klasse.

Wijzig in de LoadContent() methode de laatste regel, zie hieronder:

```
texture = Texture2D.FromStream(GraphicsDevice, textureStream)   'Oude regel

Dim texture As Texture2D = _
     Texture2D.FromStream(GraphicsDevice, textureStream)   'Nieuwe regel
```

Typ onderstaande regel eronder om de constructor van de Bal klasse aan te roepen:

```
bal = New Bal(texture)
```

Maak in de Bal klasse een nieuwe methode Update(). Hier moet de besturing in komen die nu nog in de Game1 Update() methode staat.

```
Public Sub Update()

End Sub
```

Selecteer onderstaande regels en knip en plak het in de methode:

```
spritePos += spriteSpeed * 0.01F
Dim MaxWidth As Integer = graphics.GraphicsDevice.Viewport.Width
Dim MaxHeight As Integer = graphics.GraphicsDevice.Viewport.Height

If (spritePos.X < 0) Then
     spritePos.X = 0
     spriteSpeed.X *= -1
End If
If (spritePos.X + texture.Width > MaxWidth) Then
     spritePos.X = MaxWidth - texture.Width
     spriteSpeed.X *= -1
End If
If (spritePos.Y < 0) Then
     spritePos.Y = 0
     spriteSpeed.Y *= -1
End If
If (spritePos.Y + texture.Height > MaxHeight) Then
     spritePos.Y = MaxHeight - texture.Height
     spriteSpeed.Y *= -1
End If
```

De twee graphics types voor de Viewport zijn echter onbekend. Dit komt doordat de GraphicsDeviceManager in Game1 gedefinieerd wordt. We kunnen dat niet in de Bal klasse gaan zetten, maar we willen ook de twee Max variabelen MaxWidth en MaxHeight lokaal houden. Hoe spelen we dat klaar? De oplossing is om het aangemaakte graphics object in Game1 door te geven aan de constructor, net zoals we dat hebben gedaan met het texture object.

Maak een **graphics** variabele aan bij de andere variabelen in de Bal klasse:

```
Private graphics As GraphicsDeviceManager
```

Wijzig de constructor als volgt:

```
Public Sub New(ByVal texture As Texture2D, _
               ByVal graphics As GraphicsDeviceManager)
     Me.texture = texture
     Me.graphics = graphics
End Sub
```

In de LoadContent() methode moeten we in de laatste regel het graphics object als tweede parameter meegeven:

```
bal = New Bal(texture, graphics)
```

In de Update() methode van Game1 kunnen we de Update() methode van het bal object aanroepen. Plaats de regel waar eerder de besturing stond:

```
bal.Update()
```

Als laatste hebben we de Draw() methode nodig. Maak er een in de Bal klasse zoals hieronder:

```
Public Sub Draw(ByVal spriteBatch As SpriteBatch)
     spriteBatch.Draw(texture, spritePos, Color.White)
End Sub
```

Verwijder de spriteBatch.Draw() regel in de Draw() methode van Game1 en plaats onderstaande regel op dezelfde plaats.

```
bal.Draw(spriteBatch)
```

Als u de game nu start, lijkt er niets veranderd. Maar u weet dat Game1 nu een stuk kleiner is geworden. En we nu een aparte klasse hebben gemaakt met alles erin dat voor de bal nodig is, kunt u zien hoe belangrijk het is gebruik te maken van onderdelen en niet alles in één project door elkaar te laten werken.

! In plaats van het hele graphics object door te geven aan de Bal constructor, kunnen we ook gewoon de twee variabelen MaxWidth en MaxHeight doorgeven. Dit is vooral handig als we een Basis klasse gebruiken waarvan we meerdere zoonklassen hebben, zie meer daarover in de volgende Bulletin.

Liberty BASIC API Reference.

Shell en Kernel API

Zorg ervoor dat u de sectie over het maken van API aanroepingen gelezen en begrepen hebt, voordat u verder gaat met deze sectie. Deze sectie functioneert voornamelijk als een referentie, hoewel er voorbeeld routines zijn opgenomen ter illustratie van enkele van de API-aanroepen die hier zijn vermeld.

In de volgende voorbeelden moet u ervan uitgaan dat er vensters open staan en de venster handle is “h” als teruggegeven bij de HWND functie:

```
h     =     hwnd(#main)
```

De handles van de controls beginnen met een h en beschrijven de controls. Als voorbeeld, een tekst editor gemaakt met het tekst editor commando van Liberty BASIC en aangeroepen met #main.text zal een handle met de HWND functie “hText” zijn en de handle van een graphicbox gemaakt als #main.gbox zal een handle “hGraphicBox” zijn.

```
hText          =     hwnd(#main.text)
hGraphicBox    =     hwnd(#main.gbox)
```

Om dingen zo goed mogelijk, zonder elke keer uit te hoeven leggen, te kunnen maken, zullen variabelen en parameters duidelijk te begrijpen namen krijgen, zoals dit:

```
xOrg           =     Originele X of locatie
yOrg           =     Originele Y of locatie
xExtent        =     X eindpunt
yExtent        =     Y eindpunt
width          =     breedte
```

```
height          =    hoogte
```

ExtractIcon
Deze functie haalt een pictogram van een uitvoerbaar bestand, een dll-bestand of een pictogrambestand. De “hInstance” is de instantie handle van de aangeroepen applicatie, die teruggegeven is door de GetWindowLong functie. De “szFile$” is een oneindige tekenreeks met de naam van het bestand dat het pictogram bevat. De “index” geeft de index van het pictogram die teruggegeven wordt. Als de parameter nul is, retourneert de functie de handle van het eerste pictogram in het opgegeven bestand. Als de parameter -1 is, retourneert de functie het aantal pictogrammen in het opgegeven bestand. Als de index een positieve integer is, zal de handle van het pictogram met die index teruggegeven worden. Gebruik DrawIcon om het pictogram weer te geven.

```
calldll #shell32, "ExtractIconA",_
     hInstance as ulong,_ 'instantie handle van aangeroepen applicatie
     szFile$ as ptr,_     'naam van bestand van welk pictogram te halen
     index as long,_      'index van pictogram die opgehaald moet worden
     hIcon as ulong       'handle van pictogram - gebruik met DrawIcon, _
                          '0 op mislukking
```

FindExecutable
De FindExecutable functie zoekt en retourneert de uitvoerbare bestandsnaam die gekoppeld is met een opgegeven bestandsnaam. De variabele lpszFile$ verwijst naar een oneindige tekenreeks met de opgegeven bestandsnaam. Dit kan een document of een uitvoerbaar bestand zijn. De variabele lpszDirectory$ verwijst naar een oneindige tekenreeks die de stations letter en –pad voor de standaard directory specificeert. De variabele lpszResult$ verwijst naar een buffer die de naam van een uitvoerbaar bestand bevat zodra de functie resulteert. Deze oneindige tekenreeks bevat de applicatie die gekoppeld is met de variabele lpszFile$. Onthoud dat Liberty BASIC automatisch de tekenreeksen oneindig toevoegd. Voor het toevoegen van chr$(0) is het alleen nodig als u de tekenreeks als ByRef wilt doorgeven. Een voorbeeld is:

```
lpszFile$ = fileName$

lpszDirectory$ = Directory$

lpszResult$ = space$(255) + chr$(0)

calldll #shell32, "FindExecutableA", _
     lpszFile$ as ptr, lpszDirectory$ as ptr, _
     lpszResult$ as ptr, _    'buffer om informatie te ontvangen
     result as long      'meer dan 32 als succesvol

associatedFile$ = trim$(lpszResult$)
```

Hier is een voorbeeldprogramma dat een bestand opgeeft en uitzoekt met welke applicatie het bestand te maken heeft:

```
filedialog "Open bestand..","*.*",file$
if file$="" then end

index=len(file$) : length=len(file$)

' Verdeel bestand om stations letter/pad informatie van bestandsnaam
' apart te houden
while mid$(file$, index,1)<>"\"
     index=index-1
```

```
wend

lpszFile$=right$(file$,length-index)

print "Gekozen bestand is:  " + trim$(lpszFile$)

lpszDirectory$ = left$(file$,index)

print "Station en pad zijn:  " + trim$(lpszDirectory$)

lpszResult$ = space$(255) + chr$(0)     ' maak lege buffer klaar

calldll #shell32, "FindExecutableA", _
     lpszFile$ as ptr, lpszDirectory$ as ptr, lpszResult$ as ptr, _
     result as long      'meer dan 32 als succesvol

associatedFile$ = trim$(lpszResult$)

print "Gekoppeld bestand is:  " + associatedFile$
'    Uitvoer van dit programma kan zijn:
'    Gekozen bestand is:  NEW142.TXT
'    Station en pad zijn:  C:\LB142W\
'    Gekoppeld bestand is:  C:\WINDOWS\NOTEPAD.EXE
```

GetDiskFreeSpaceEx

De GetDiskFreeSpaceExA functie haalt informatie over het opgegeven station, inclusief de vrije ruimte op de schijf. De teruggave is niet-nul als de functie slaagt. Vanwege de functie pointers vereist naar numerieke variabelen als argumenten, worden ze doorgegeven als structs. De waarden moeten grote integers zijn. In Liberty BASIC hebben de structs ieder twee members; de ene voor de lage waarde en de tweede voor de hoge waarde. De reden dat dit nodig is, komt omdat Liberty BASIC geen type kent voor grote integers. Nadat de functie teruggegeven heeft, kunnen de waarden weer vanuit de structs worden opgehaald. Om het aantal bytes op te halen, moeten de waarden vanuit de structs vermenigvuldigd worden en dan aan elkaar toegevoegd worden, zoals onderstaande demo laat zien.

```
dir$="c:\"
struct AB, _
     freebytesLo as ulong, _
     freebytesHi as ulong         'vrije bytes beschikbaar
struct TB, _
     totalbytesLo as ulong, _
     totalbytesHi as ulong        'totaal bytes
struct FB, _
     totalfreeLo as ulong, _
     totalfreeHi as ulong         'totaal vrije bytes

calldll #kernel32, "GetDiskFreeSpaceExA", _
     dir$ as ptr, _       'directory naam
     AB as struct, _      'vrije bytes op schijf beschikbaar voor de
                          'aanroeper
     TB as struct, _      'aantal bytes op schijf
     FB as struct, _      'totaal vrije bytes op schijf
     ret as boolean       'niet-nul=succes

avBytes = AB.freebytesHi.struct * _
          HexDec("100000000") + AB.freebytesLo.struct
```

```
print "Totaal beschikbare bytes is "; avBytes
print "Totaal beschikbare KB is ongeveer "; int(avBytes/1024)
print "Totaal beschikbare MB is ongeveer "; int((avBytes/1024)/1024)
print "Totaal beschikbare GB is ongeveer "; int(((avBytes/1024)/1024)/1024)
print

totalBytes = TB.totalbytesHi.struct * _
          HexDec("100000000") + TB.totalbytesLo.struct
print "Totale bytes is "; totalBytes
print "Totale KB is ongeveer "; int(totalBytes/1024)
print "Totale MB is ongeveer "; int((totalBytes/1024)/1024)
print "Totale GB is ongeveer "; int(((totalBytes/1024)/1024)/1024)
print

freeBytes = FB.totalfreeHi.struct * _
          HexDec("100000000") + FB.totalfreeLo.struct
print "Totale vrije bytes is "; freeBytes
print "Totale vrije KB is ongeveer "; int(freeBytes/1024)
print "Totale vrije MB is ongeveer "; int((freeBytes/1024)/1024)
print "Totale vrije GB is ongeveer "; int(((freeBytes/1024)/1024)/1024)
print

usedkb = int((totalBytes - freeBytes)/1024)
print "Totale KB gebruikt is ongeveer "; usedkb
print "Totale MB gebruikt is ongeveer "; int(usedkb/1024)
print "Totale GB gebruikt is ongeveer "; int((usedkb/1024)/1024)
print

if ret=0 then print "Fout gegaan" else print "Succes"
end
```

GetDriveType

Deze functie bepaald of het opgegeven station, een vaste of een netwerk (extern) station, verwijderbaar is. Onthoud dat de Liberty BASIC Drive$ variabele een lijst van stations vasthoud, maar niet eindigen met backslash es. Het resulteert de stations in de vorm: “a: c: d:” als gebruikte informatie van de teruggegeven tekenreeks en zorg ervoor de laatste backslash toe te voegen voor het als argument aan deze API functie door te geven.

```
drivename$="c:\"    'vereist een laatste backslash
calldll #kernel32, "GetDriveTypeA", _
     drivename$ as ptr, _     'tekenreeks als inhoud de naam van de schijf
     driveType as long        'resulteert code voor station type

0                  Het station type kan niet worden bepaald.
1                  De root directory bestaat niet.
_DRIVE_REMOVABLE=2 De schijf kan worden verwijderd van het station.
_DRIVE_FIXED=3     De schijf kan niet worden verwijderd van het station.
_DRIVE_REMOTE=4    Het station is een externe (netwerk) schijf.
_DRIVE_CDROM=5     Het station is een CD-ROM station
_DRIVE_RAMDISK=6   Het station is een RAM schijf
```

GetLogicalDriveStrings

Deze functie vereist een tekenreeks buffer om de teruggegeven informatie over de huidige, overige, stations vast te houden. Roep de functie één keer met een zeer smalle tekenreeks buffer om de grootte van de buffer, die nodig is, op te vragen, dan maak een buffer van juiste grootte en roep de functie

nog eens aan. De teruggekregen tekenreeks bevat een lijst van alle stations, gescheiden door null karakters. Ontleed de lijst zoals in de demo om de individuele stations letters te krijgen. De Drive$ variabele van Liberty BASIC is gevuld wanneer het programma start en niet veranderd, zelfs als de gebruiker een station toevoegd of verwijderd, zoals een geheugen stick.

```
driveString$ = space$(2) 'zet een smalle tekenreeks buffer op
lenBuffer = len(driveString$)
'maak de API aanroep om uit te vinden hoe groot de buffer moet zijn
calldll #kernel32, "GetLogicalDriveStringsA", _
     lenBuffer as ulong, _    'lengte van buffer
     driveString$ as ptr, _   'tekenreeks buffer zal worden gevuld door
                              'functie
     numBytes as ulong        'retourneert de benodigde buffer grootte

driveString$ = space$(numBytes+1) 'vergroot buffer voor het houden van
                                  'de verkregen informatie
lenBuffer = len(driveString$)

'maak de aanroep nog eens om de buffer te vullen met stations informatie
calldll #kernel32, "GetLogicalDriveStringsA", _
     lenBuffer as ulong, _    'lengte van buffer
     driveString$ as ptr, _   'tekenreeks buffer zal worden gevuld door
                              'functie
     numBytes as ulong        'verkregen lengte van de tekenreeks

'de driveString$ bevat nu een aantal stations, gescheiden door null
'karakters
d$ = left$(driveString$,numBytes)
print "API verkregen driveString$ is ";chr$(34);d$;chr$(34)

'breek nu in de individuele stations informatie:
print "Huidige stations op deze machine zijn als volgt:"
print
i = 1     'start met eerste station in de tekenreeks
while word$(d$,i)<>""
     print word$(d$,i)
     i = i + 1
wend
```

GetModuleFileName
Deze functie haalt de bestandsnaam van een module die vervolgens kan worden gebruikt voor aanmelding van de programma's die worden uitgevoerd. Haal de “task” handle met LoadLibrary om in deze functie door te voeren.

```
File$ = space$(255) + chr$(0)      'buffer om bestandsnaam te ontvangen

Length = len(File$)                'buffer grootte

calldll #kernel32, "GetModuleFileNameA", _
     task as ulong, _          'task handle vanuit LoadLibrary
     File$ as ptr, _           'buffer om bestandsnaam te ontvangen
     Length as ulong, _        'buffer grootte
     result as ulong
```

File$ bevat nu de module bestandsnaam.

GetModuleHandle
Deze functie haalt de handle van de opgegeven module. Het resultaat is 0 als het programma niet wordt uitgevoerd, anders wordt het de module handle. Als de name parameter is NULL of gelijk aan 0, geeft GetModuleHandle als resultaat een handle van het bestand gebruikt voor het maken van het aanroepende proces.

```
calldll #kernel32, "GetModuleHandleA", _
     "kernel32" as ptr, _      'naam van EXE of DLL om te controleren
     result as ulong           '0 = niet uitgevoerd,
                               'groter dan 0 = wordt uitgevoerd
```

of

```
calldll #kernel32, "GetModuleHandleA", _
     0 as long, _          'null = geef module handle van dit programma
     result as ulong       'module handle van dit programma
```

GetProfileString
De GetProfileString functie haalt de tekenreeks geassocieerd met een ingang binnen de opgegeven sectie in het WIN.INI initialisatie bestand. Als de functie succesvol is, resulteert het een aantal bytes gekopieerd naar de buffer, exclusief de afsluitende nul. Als “keyName$” is “device” zal de functie de buffer vullen met de naam van de standaard printer die op het systeem geïnstalleerd is. Als “keyNa-me$” is “” dan zal de functie de buffer “result$” vullen met een tekenreeks dat uit alle ingangen in de gespecificeerde sectie bestaat, in dit geval “windows”. Voorbeeldprogramma.

```
appName$ = "windows"
keyName$ = "device"
default$ = ""
size = 250
result$ = space$(size)+chr$(0)

calldll #kernel32, "GetProfileStringA", _
     appName$ as ptr, _   'windows
     keyName$ as ptr, _   'device wordt bedoeld printer
     default$ as ptr, _   'null, geen waarde
     result$ as ptr, _    'buffer die informatie bevat
     size as long, _      'buffer grootte
     result as long       'grootte van resulterende tekenreeks

'geef de opgehaalde informatie weer, maar niet met de afsluitende ASCII nul
print left$(result$, instr(result$, chr$(0)) - 1)
```

Wordt vervolgd

PowerBASIC – Gebruiker-gedefinieerde types.

Het definiëren en gebruiken van gebruiker-gedefinieerde types

Waar het in dit onderwerp van PowerBASIC om gaat, zijn de gebruiker-gedefinieerde types. In de meeste BASIC programmeertalen kan dat ook. PowerBASIC heeft echter een mogelijkheid om de tekstlengten van stringvelden tijdens de uitvoer te kunnen wijzigen.

De definitie van een gebruiker-gedefinieerd type begint met het sleutelwoord TYPE en eindigt met de sleutelwoorden END TYPE. Tussen deze sleutelwoorden definieert u de namen en de gegevenstypes van de elementen (velden) die tot een deel van het nieuwe type behoren. Bijvoorbeeld:

```
TYPE StudentRecord
     LastName AS STRING * 20       ' Een tekenreeks van 20 karakters
     FirstName AS STRING * 15      ' Een tekenreeks van 15 karakters
     IDnum AS LONG                 ' Student ID, een grote integer
     Contact AS STRING * 30        ' Contactpersoon voor noodsituaties
     ContactPhone AS STRING * 14   ' Het telefoonnummer
     ContactRel AS STRING * 8      ' Relatie van de contactpersoon tot
                                   ' de student
     AverageGrade AS SINGLE        ' Enkele-precisie percentage niveau
END TYPE
```

Onthoud goed dat de definitie van een gebruiker-gedefinieerd type geen geheugen opzij legt voor het opslaan van gegevens van dat type. Anders gezegd, het definieert een sjabloon voor het nieuwe type *StudentRecord*. Komt de compiler een statement tegen voor het declareren (of maken) van een variabele van het nieuwe type, dan zal die willen “weten” hoeveel bytes geheugenruimte opzij gehouden moeten worden voor de variabele. Hieronder ziet u een voorbeeld hoe de variabele gedeclareerd wordt met het DIM statement:

```
DIM Student AS StudentRecord
```

Toegang tot de velden van een gebruiker-gedefinieerd type

Om met de velden te kunnen werken binnen een record variabele, moet de veldnaam met de variabele naam gescheiden worden met een punt. Hier zijn wat voorbeelden met het gebruik van de *Student* variabele die hierboven door het DIM statement gedeclareerd is:

```
PRINT Student.LastName
PRINT "Id number is: "; Student.IdNum
Student.FirstName = "Bob"
Student.LastName = "Smith"
PRINT Student.LastName; ", "; Student.FirstName
```

Hoewel u nog steeds in PowerBASIC punten kunt gebruiken binnen eenvoudige variabelennamen, zou het een goed idee zijn om het gebruik van de punt te beperken voor de toegang tot de variabele recordvelden. PowerBASIC kan controleren om erachter te komen wat u bedoelt met *User.Id*, maar wat als de mensen uw programma gaan lezen? Is het een veld binnen een record variabele genaamd *User* of een simpele variabele genaamd *User.Id*? Waarom de dingen ingewikkelder maken dan ze nodig zijn?

Gebruiker-gedefinieerde types nesten

De velden binnen een gebruiker-gedefinieerd type kan worden opgemaakt met andere gebruiker-gedefinieerde types. Net als een aantal Chinese dozen, met elke doos die als inhoud een kleinere doos heeft – u kunt een gebruiker-gedefinieerd type nesten binnen de ander. Het eindresultaat is dat u gegevensstructuren maakt die een hiërarchie heeft vergelijkbaar als de mappen van uw harde schijf.

In plaats daarvan kunnen we de twee aparte studentnamen velden als een apart *NameRec* type maken:

```
TYPE NameRec
     Last AS STRING * 20
```

```
     First AS STRING * 15
     Initial AS STRING * 1
END TYPE
```

Wanneer we het *Student Record* type definiëren, kunnen we de veldnaam defniëren met *NameRec*:

```
TYPE StudentRecord
     FullName AS NameRec
     IdNum AS LONG
     Contact AS NameRec
     ContactPhone AS STRING * 14
     ContactRel AS STRING * 8
     AverageGrade AS SINGLE
END TYPE
```

U kunt natuurlijk nog een stap verder gaan en andere componenten definiëren van het student record als geneste records. Bijvoorbeeld, een *ContactRecord* of zelfs een *PhoneRec*, maar dat is geheel aan u wat u het beste vindt. Om toegang te krijgen tot de velden van een geneste record, maakt u gebruik van de punt notatie. Net zoals de backslash (\) wordt gebruikt om sub-mapnamen in een pad (BV, C:\PB\PROGRAM) te scheiden, wordt de punt gebruikt om binnen de record variabelennamen de veldelementen te scheiden, bijvoorbeeld:

```
StudentRecord.FullName
```

verwijst naar het *FullName* veld (dat van het type *NameRec* is) binnen *Student Record*, en

```
StudentRecord.FullName.First
```

verwijst naar het sub-veld *First* binnen het *FullName* veld.

U kunt gebruiker-gedefinieerde types nesten zo diep nesten als u wilt zolang de gehele naam, die wordt gebruikt om te verwijzen naar een veld binnen de identifier, een maximale lengte van 255 tekens is. In de praktijk echter zou u waarschijnlijk maar twee, hooguit drie niveaus nesten. Verder wordt het onhandig, is het lastig te onthouden, en je hebt meer kans om typefouten maken.

Arrays gebruiken van gebruiker-gedefinieerde types

U kunt arrays maken van gebruiker-gedefinieerde types net zoals u arrays kunt maken van integers of strings of elk ander PowerBASIC gegevenstype. Bijvoorbeeld:

```
DIM Class(1:30) AS StudentRecord
```

Om toegang te krijgen tot verschillende elementen van de *Class* array, gebruikt u subscripts net zoals u met elk ander array doet. Het derde student record is bijvoorbeeld *Class(3)*. Het punt scheidingsteken en de veldnaam volgt na het array subscript:

```
PRINT Class(3).FullName.First
```

zal de waarde van *first* van het derde student in de class array printen. Denk eraan: het array is gemaakt van elementen van het type *StudentRecord*, dus het subscript behoort tot de variabele naam als één geheel.

U kunt multidimensionale arrays maken van gebruiker-gedefinieerde types net zoals u met elk ander PowerBASIC type kunt doen. De grens van het aantal elementen en dimensies in dergelijke arrays beheerst dezelfde regels, evenals de grenzen worden bepaald door de hoeveelheid opslag van de gegevens, vereist voor elk element.

Gebruiker-gedefinieerde types met procedures en functies gebruiken

Procedures en functies kunnen gebruiker-gedefinieerde types verwerken als elk ander gegevenstype. Deze sectie behandelt de volgende onderwerpen:

- Velden doorgeven als argumenten
- Records doorgeven als argumenten
- Record arrays doorgeven als argumenten

Velden doorgeven als argumenten

Velden in een gebruiker-gedefinieerd type die als PowerBASIC types (INTEGER, WORD, STRING enzovoort) zijn opgebouwd, worden doorgegeven aan procedures en functies als simpele variabelen. Bijvoorbeeld, gegeven het volgend gebruiker-gedefinieerd type *PatientRecord* is als volgt:

```
TYPE PatientRecord
     FullName AS STRING * 32
     AmountDue AS DOUBLE
     IdNum AS LONG
END TYPE
DIM Patient AS PatientRecord
```

u kunt een procedure gebruiken genaamd *PrintStatement*

```
SUB PrintStatement(Id AS LONG, AmountPastDue AS DOUBLE)
     ' access Id and AmountPastDue
     ...
END SUB
```

en aanroepen als:

```
CALL PrintStatement(Patient.IdNum, Patient.AmountDue)
```

Records doorgeven als argumenten

U kunt ook uw procedures en functies schrijven om argumenten te accepteren van gebruiker-gedefinieerde types. Dit is nuttig als u meerdere argumenten moet doorgeven; in plaats van een lange lijst argumenten kunt u een enkel gebruiker-gedefinieerd type doorgeven. Bijvoorbeeld, gegeven het *PatientRecord* gebruikerstype van de vorige sectie, kunt u uw *PrintStatement* procedure als volgt schrijven:

```
SUB PrintStatement(Patient AS PatientRecord)
     ' access Patient.IdNum and Patient.AmountDue
     ...
END SUB
```

Nu kunt u *PrintStatement* aanroepen als:

```
CALL PrintStatement(Patient)
```

Record arrays doorgeven als argumenten

Procedures en functies accepteren arrays van records net zo gemakkelijk als ze arrays kunnen accepteren van andere types. Bijvoorbeeld, als u een array heeft van *PatientRecords,* met elk verschuldigd bedrag inhoud in een patiënt record, dan kunt u een functie schrijven die het totaal verschuldigd bedrag van alle patiënt records in de array resulteert:

```
FUNCTION TotalAmountDue(Patients() AS PatientRecord) AS DOUBLE
```

```
     DIM total AS DOUBLE

     total = 0
     FOR i = LBOUND(Patients) TO UBOUND(Patients)
          total = total + Patients(i).AmountDue
     NEXT

     TotalAmountDue = total
END FUNCTION
```

U kunt de functie aanroepen als dit:

```
DIM Patients(1:100) AS PatientRecord
...
PRINT "Total amount due:"; TotalAmountDue(Patients())
```

Opslag benodigdheden

U kunt de hoeveelheid opslagruimte bepalen die nodig is voor een variabele van een gebruiker-gedefinieerd type met de LEN functie. U moet echter de variabele van het type doorgeven, niet de naam van het type zelf (als gedefinieerd in het TYPE statement). Vergeet niet dat de naam in het TYPE statement alleen maar gebruikt wordt als een sjabloon. Om bijvoorbeeld de benodigdheden voor een student record te bepalen, gebruik dan:

```
RecordSize = LEN(Student)
```

Om de opslag benodigdheden voor een dynamische array van *Student Records* te bepalen:

```
DO
     INPUT "How many records do you want: "; NumRecords
     RequiredMemory = LEN(Student) * NumRecords
     IF RequiredMemory > FRE(0) THEN
          PRINT "Not enough memory to create the array!"
          PRINT "Please enter a smaller number."
          NumRecords = 0
     END IF
LOOP UNTIL NumRecords > 0
```

Het adres van een record variabele, zoals terug wordt gegeven door de VARPTR functie, is het eerste gegevensbyte adres in het geheugen in de record. Ook kunt u het beginadres van de velden in de record, door de volledige naam van het veld (*Student.IdNum*, als voorbeeld), doorgeven aan de functie VARPTR.

Unions

Als u ooit eens geprogrammeerd heeft in Pascal of C, dan is het mogelijk dat u vertrouwd bent met het concept van een union (in het Nederlands genoemd: unie). In sommige gevallen is een union hetzelfde als een gebruiker-gedefinieerd type, inbegrepen met records en andere unions, met uitzondering van het UNION sleutelwoord worden ze op dezelfde manier gedefinieerd. Het grote verschil tussen de gebruiker-gedefinieerde typen en unions is dat elk veld binnen een union dezelfde geheugenplaats van alle anderen bezet. Laten we eens een voorbeeld bekijken. De volgende definitie zal een union maken met als naam *Location* en een *Location* variabele met als naam *ScreenLoc*:

```
TYPE HiLo
     Hi AS BYTE
     Lo AS BYTE
```

```
END TYPE

UNION Location
     WordFld AS WORD
     ByteFld AS HiLo
END UNION

DIM ScreenLoc AS Location

INPUT "Enter a value for ScreenLoc: ", ScreenLoc.WordFld
PRINT "The value of WordFld is:"; ScreenLoc.WordFld
PRINT "The value of the high byte is:"; ScreenLoc.ByteFld.Hi
PRINT "and the value of the low byte is:"; ScreenLoc.ByteFld.Lo
```

Wanneer u gebruik maakt van het veld *ScreenLoc.WordFld*, leest u de hele inhoud van de union als een integer. Wanneer u verwijst naar *ScreenLoc.ByteFld.Hi*, wordt u verwezen naar de hoge byte van *ScreenLoc*.

Dynamische structuren met flex strings maken

Gebruiker-gedefinieerde types en unions (TYPE en UNION) zijn erg handig voor de meeste gegevensstructuren die nodig zijn. Doordat ze gedefinieerd worden tijdens de compileer-tijd, zijn ze heel snel voor gebruik, en u kunt ze gemakkelijk gebruiken om random-access bestanden of binaire bestanden in te lezen en weg te schrijven.

Echter wilt u soms meer flexibiliteit dan types en unions kunnen geven. Doordat ze gedefinieerd worden tijdens de compileer-tijd, moet u de grootte opgeven als een constante (of een letterlijke waarde). Hieronder ziet u nogmaals het type StudentRecord:

```
TYPE StudentRecord
     LastName AS STRING * 20       ' Een tekenreeks van 20 karakters
     FirstName AS STRING * 15      ' Een tekenreeks van 15 karakters
     IDnum AS LONG                 ' Student ID, een grote integer
     Contact AS STRING * 30        ' Contactpersoon voor noodsituaties
     ContactPhone AS STRING * 14   ' Het telefoonnummer
     ContactRel AS STRING * 8      ' Relatie van de contactpersoon tot
                                   ' de student
     AverageGrade AS SINGLE        ' Enkele-precisie percentage niveau
END TYPE
```

Dit type zal voor de meeste doeleinden volstaan, maar wat als er aanpassingen nodig zijn? U kunt geen gebruiker-gedefinieerd type aanpassen tijdens de uitvoertijd; zodra u het compileert is het gemaakt. Hoewel, u kunt wel flex strings aanpassen.

Flex strings aanpassen

Stel, u wilt dat de gebruiker zelf de lengte aanpast van elke member in de gegevensstructuur. U hebt dan de code zoals dit:

```
configFile = freefile

OPEN "CONFIG.DAT" FOR INPUT AS #configFile
     INPUT #configFile,  lenStudentLastName
     INPUT #configFile,  lenStudentFirstName
     INPUT #configFile,  lenStudentContact
     INPUT #configFile,  lenStudentContactPhone
```

```
     INPUT #configFile,  lenStudentContactRel
CLOSE #configFile
```

Dan gebruikt u deze waarden om een flex string structuur te bouwen:

```
MAP  StudentFlex$$ *     lenStudentLastName + _
                         lenStudentFirstName + _
                         4 + _
                         lenStudentContact + _
                         lenStudentContactPhone + _
                         lenStudentContactRel + _
                         4, _
     lenStudentLastName AS StudentLastName$$, _
     lenStudentFirstName AS StudentFirstName$$, _
     4 AS StudentIDnum$$, _
     lenStudentContact AS StudentContact$$, _
     lenStudentContactPhone AS StudentContactPhone$$, _
     lenStudentContactRel AS StudentContactRel$$, _
     4 AS StudentAverageGrade$$

studentFile = freefile
OPEN "STUDENTS.DAT" FOR INPUT AS #studentFile

DO UNTIL EOF(studentFile)
     INPUT #studentFile, StudentFlex$$
     PRINT RTRIM$(StudentFirstName$$); " ";
     PRINT RTRIM$(StudentLastName$$); " ";
     PRINT "(ID #"; CVL(StudentIDnum$$);
     PRINT ") has an average grade of ";
     PRINT CVS(StudentAverageGrade$$);
LOOP
```

Nadelen van de flex strings

Er zijn twee grote nadelen bij het gebruik van flex strings:

- *Snelheid*. Omdat flex strings gedefinieerd zijn tijdens de uitvoertijd, is er een kleine verwerkingstijd betrokken. Gebruiker-gedefinieerde types zijn statisch – gedefinieerd in compileertijd – zodat er geen extra overschot is.
- *Gemakkelijk in gebruik*. Flex strings moeten gedefinieerd worden met gebruik van het MAP statement. Het definiëren van een flex string structuur is gecompliceerder dan een gebruiker-gedefinieerd type. Ook bestaat een flex string structuur alleen uit flex strings, dus om een numerieke waarde in te voegen moet u gebruik maken van string-naar-numeriek en numeriek-naar-string converteer functies (respectievelijk *CVx* en *MKx$*). U kunt direct numerieke waarden aan numerieke members van een gebruiker-gedefinieerd type toekennen.

Excel – De functies op het werkblad.

De functies die u in de vorige Bulletin zag staan, kunt u, zoals u zag, kiezen uit categorieën. Als u echter de namen van de functies kent, hoeft u niet speciaal naar de functies te zoeken. U kunt ze ook direct in een cel opgeven.

Het verschil tussen een functie en een formule

Een functie en een formule kunnen we als volgt omschrijven: In een functie wordt een formule bere-

kend (expressie), maar een formule op zich is een directe opgegeven expressie zonder naam. Onderstaande twee voorbeelden doen hetzelfde, maar de ene is een functie en de andere een formule:

Functie: =SOM(A1:A3)
Formule: =A1+A2+A3

Het voordeel van een functie is dat de expressie niet te zien is, alleen de parameter. De SOM functie vraagt om een bereik of om een berekening. Overal waar we een som functie gebruiken, wordt er automatisch een expressie gemaakt (uitgezonderd een berekening) en uitgevoerd.

Geven we onderstaande som op:

=SOM(2+3+4)

Dan geeft Excel als resultaat een 9.

Normaal gesproken telt de SOM alles op wat in een bereik ligt, maar de SOM accepteert in een berekening ook andere prioriteiten, zoals vermenigvuldigen en delen. In principe is de SOM functie daarvoor helemaal niet nodig, want we kunnen in een cel ook typen:

=2+3+4

maar niet:

=A5:A7

Excel geeft dan meteen de foutmelding: #WAARDE!, omdat Excel niet weet wat hij met een bereik moet doen. Alleen als een bereik als een parameter in een functie wordt gebruikt, heeft het een doel.

De functies in Excel kunnen heel veel. Ze zijn erg nuttig in boekhoudingen en laboratoria. Deze functies bestaan niet in VBA, maar kunnen wel aangeroepen worden via het werkblad object om de functie op te roepen. Het is niet gemakkelijk om het handmatig in VBA te gebruiken, en dan is het maar beter om op dat moment even een macro op te nemen.

Logica en sub-functies

Staat er in cel A1 de waarde 10 en geven we in cel A2 de formule =A1>5, dan krijgen we als antwoord: WAAR. Excel kent dezelfde logische operatoren als BASIC.

In plaats van alleen maar WAAR of ONWAAR te krijgen, kunnen we ook eigen tekst weergeven uit bepaalde logica. Excel kent een keuzestructuur die we ook in VBA kunnen gebruiken. De ALS functie wordt daarvoor gebruikt:

=ALS(A1>5;”Hallo”;”Niet hallo”)

Nu zal niet de waarde WAAR worden weergegeven, maar de tekst *Hallo*. Houd er rekening mee om de aanhalingstekens te gebruiken.
Hebben we ook een getal in cel B1 staan, dan kunnen we A1 en B1 samen vergelijken:

=ALS(A1<>B1;”Niet gelijk”;”Gelijk”)

We kunnen de ALS functie vergelijken met de IIF functie in BASIC:

```
n$ = IIF(A1>5,"Hallo","Niet hallo")
```

Staat er in VBA echter onderstaande conditie in de functie, dan hebben we een probleem:

```
C1 = IIF(A1<>B1 AND A1>100,A1,B1)
```

Excel kent geen operatoren als AND en OR en ook niet door het in het Nederlands te proberen. Op het werkblad kunnen we maar per één cel controleren, maar niet of de ene cel én nog eens de andere cel aan een bepaalde waarde zijn voldaan.

Om toch iets van een AND operator na te bootsen, zouden we de ALS functie kunnen nesten. Onderstaande regel kan lastig zijn om te begrijpen:

=ALS(A1<>B1;ALS(A1>100;C1;D1);B1)

Als we dit omzetten in VBA, dan staat er eigenlijk:

```
n = IIF(A1<>B1 AND A1>100,C1,D1)
```

Alleen C1 kan gebruikt worden als zowel A1 ongelijk is aan B1 en A1 groter is dan 100. Omdat we in Excel de ALS functie voor een soort AND operator een tweede ALS functie moeten nesten, ontstaan er twee ELSE onwaar parameters. Vandaar dat u cel B1 in de IIF functie mist.

Voor een OR operator zouden we de ALS functie moeten nesten in de ELSE onwaar parameter, maar dan krijgen we weer de situatie dat we een ELSE teveel hebben.

Om dezelfde uitvoer te krijgen als bovenstaande ALS functie, moeten we de IIF functie in VB ook nesten:

```
n = IIF(A1<>B1,IIF(A1>100,C1,D1),B1)
```

Nu hebben we wel twee ELSE onwaar situaties, omdat we nu nesten en geen gebruik maken van een conditie met een logische operator.

Een ander probleem met functies en bereiken is dat we geen controle in een cel uit kunnen voeren in de actieve cel. Excel zal dan een kringverwijzingsfoutmelding geven. De functies moeten we dus buiten het bereik in een andere cel uitvoeren om fouten te voorkomen. In VBA maakt dat niet uit, omdat we dan gebruik maken van variabelen.

Willen we de ALS functie laten doorlopen in een optelling, dus in een som-bereik, dan moeten we in VBA een lus gebruiken. Op het werkblad is dat niet nodig; we hoeven dan ook geen macro te maken of op te nemen.

Op het werkblad bestaat er een SOM functie met een sub-functie ALS. Een sub-functie is niet hetzelfde als een geneste functie. Een sub-functie wordt gescheiden met een punt.

Op de volgende pagina kunt u een voorbeeld zien hoe de SOM functie samen met de ALS functie werkt. Dit geeft veel functionaliteit voor het bepalen van berekeningen en zelfs voor het optellen van de waarden als aan bepaalde tekstwaarden zijn voldaan (zie later meer).

Het verschil tussen een sub-functie in een som en een normale som.

	A	B	C	D
1	10			
2	20			
3	30			
4	20			
5	10			
6	50			
7	40			
8	30			
9	20			
10	10			
11	90	=SOM.ALS(A1:A10;"<30")		
12	240	=SOM(A1:A10)		
13				
14				
15				

In rij 11 ziet u dat alleen de waarden, die kleiner zijn dan 30, opgeteld mogen worden. Het resultaat is dan ook 90.

De normale som berekent het hele bereik zonder controle. Dit geeft een enorm verschil. Daarom is een SOM.ALS functie zeer nuttig voor het doorzoeken van de juiste criteria.

De SOM.ALS functie kent echter nog een derde parameter: het optelbereik.
Deze is optioneel en kan worden gebruikt als niet het bereik opgeteld moet worden waar het criterium in staat. Het voorbeeld rechts geeft het eerste voorbeeld uitgebreider weer.

	A	B	C	D	E
1	10	10			
2	20	20			
3	30	30			
4	5	20			
5	95	10			
6	50	50			
7	40	40			
8	30	30			
9	70	20			
10	5	10			
11	90	=SOM.ALS(B1:B10;"<30")			
12	205	=SOM.ALS(B1:B10;"<30";A1:A10)			
13					
14					
15					

In rij 11 ziet u hetzelfde resultaat uit het bereik van kolom B, maar in rij 12 wordt nu het optelbereik van kolom A als derde parameter opgegeven. Wat er nu gebeurd is als volgt:

- Er wordt weer gecontroleerd of in het bereik de waarde kleiner is dan 30.
- Is dat het geval, dan zal de waarde in dezelfde rij, maar in de kolom van het optelbereik, opgeteld worden.
- De volgende cellen worden dus meegenomen: A1, A2, A4, A5, A9, A10

	A	B	C	D	E
1	Klaas	10			
2	Jan	20			
3	Peter	30			
4	Jan	20			
5	Kees	10			
6	Peter	50			
7	Jan	40			
8	Ineke	30			
9	Darlene	20			
10	Jan	10			
11	90	=SOM.ALS(A1:A10;"Jan";B1:B10)			
12					
13					
14					
15					

Het laatste voorbeeld links laat zien dat we ook naar personen kunnen zoeken en de waarden uit een ander bereik op kunnen tellen. Dit werkt op dezelfde manier, en om waarden van Jan te kunnen vinden is dit zeer nuttig. Daarom is de SOM.ALS functie bijzonder krachtig en zeer functioneel.

Excel kent nog meer functies die weinig bekend zijn. De volgende keer zal ik nog meer functies uit de categorieën halen en laten zien.

Cursussen

Liberty BASIC:
Cursus en naslagwerk, beide met voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

Qbasic:
Cursus, lesmateriaal en voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

QuickBasic:
Cursusboek en het lesvoorbeeld op diskette, € 11,00 voor leden. Niet leden € 13,50.

Visual Basic 6.0:
Cursus, lesmateriaal en voorbeelden op CD-ROM, € 6,00 voor leden. Niet leden € 10,00.

Basiscursus voor senioren, Windows 95/98,
Word 97 en internet voor senioren, (geen diskette). € 11,00 voor leden. Niet leden € 13,50.

Computercursus voor iedereen: tekstverwerking met Office en eventueel met VBA, Internet en programmeertalen, waaronder ook Basic, die u zou willen leren.
Elke dinsdag, woendag en vrijdag in buurthuis Bronveld in Barneveld van 19:00 uur tot 21:00 uur op de dinsdag en van 9:00 uur tot 11:00 uur op de woensdag en vrijdag. Kosten € 5,00 per week.
Meer informatie? Kijk op 'http://www.i-t-s.nl/rdkcomputerservice/index.php' of neem contact op met mij.

Computerworkshop voor iedereen; heeft u vragen over tekstverwerking of BASIC, dan kunt u elke 2de en 4de week per maand terecht in hetzelfde buurthuis Bronveld in Barneveld van 19:15 uur tot 21:15 uur. Kosten € 5,00.
Meer informatie? Kijk op ‘http://www.buurthuisbronveld.nl’ of neem contact op met mij. Voor overige informatie: http://www.tronicasoftware.nl

Software

Catalogusdiskette,	€ 1,40 voor leden. Niet leden € 2,50.
Overige diskettes,	€ 3,40 voor leden. Niet leden € 4,50.
CD-ROM’s,	€ 9,50 voor leden. Niet leden € 12,50.

Hoe te bestellen

De cursussen, diskettes of CD-ROM kunnen worden besteld door het sturen van een e-mail naar penm@basic-gg.hcc.nl en storting van het verschuldigde bedrag op:

ABN-AMRO nummer 49.57.40.314
HCC BASIC ig
Haarlem

Onder vermelding van het gewenste artikel. Vermeld in elk geval in uw e-mail ook uw adres aangezien dit bij elektronisch bankieren niet wordt meegezonden. Houd rekening met een leveringstijd van ca. 2 weken.
Teksten en broncodes van de nieuwsbrieven zijn te downloaden vanaf onze website (http://www.basic.hccnet.nl). De diskettes worden bij tijd en wijlen aangevuld met bruikbare hulp- en voorbeeldprogramma's.
Op de catalogusdiskette staat een korte maar duidelijke beschrijving van elk programma.

Alle prijzen zijn inclusief verzendkosten voor Nederland en België.

De volgende personen zijn op de aangegeven tijden beschikbaar voor vragen over programmeerproblemen. Respecteer hun privé-leven en bel alstublieft alleen op de aangegeven tijden.

Waarover	Wie	Wanneer	Tijd	Telefoon	Email
Liberty BASIC	Gordon Rahman	ma. t/m zo.	19-23	(023) 5334881	grahman@planet.nl
MSX-Basic	Erwin Nicolai	vr. t/m zo.	18-22	(0516) 541680	basic@lordthanatos.com
PowerBasic CC	Fred Luchsinger	ma. t/m vr.	19-21		f.luchsinger@kader.hcc.nl
QBasic, QuickBasic	Jan v.d. Linden				j.vd.linden@kader.hcc.nl
Visual Basic voor Windows	Jeroen v. Hezik	ma. t/m zo.	19-21	(0346) 214131	j.a.van.hezik@kader.hcc.nl
Visual Basic .NET	Marco Kurvers	vr. t/m zo.	19-22	06 30896598	m.a.kurvers@live.nl
Basic algemeen, zoals VBA Office en XNA Game Studio	Marco Kurvers	vr. t/m zo.	19-22	06 30896598	m.a.kurvers@live.nl