Basic Bulletin

21ste jaargang december 2014 Nummer 4

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

Fouten worden vaak gemaakt tijdens het programmeren. Er bestaan twee soorten: compilerfoutmeldingen en runtime foutmeldingen, of te wel uitvoerbare foutmeldingen. De laatste is het meest voorkomende en veroorzaakt daarom vaak de schoonheidsfoutjes die moeilijk te vinden zijn.

Tekstdelen uit een complete string halen is niet altijd gemakkelijk. Vooral niet als we niet eens weten op welke plek we een tekstdeel kunnen knippen. Gegevens die als één regel waarden bij elkaar opgeslagen worden, moeten ook weer ontleden kunnen worden. Een voorbeeld hoe dat gedaan wordt met dagdelen, kunt u hier vinden.

Programmeertip!
Kijk voor meer informatie over programmeren op mijn website www.tronicasoftware.nl . Nog niet alles is af, maar er zijn al wat pagina's aanwezig die u kunt bekijken. Mijn website zal telkens vernieuwd worden met steeds weer nieuwe pagina's erbij.

Marco Kurvers

BBC BASIC for Windows – De library's (3).

Volgbalken en voortgangsbalken.

Deze balken, de volgbalken en vooruitgang balken of ook genoemd *trackbars en progress bars*, staan in de **WINLIB3** library met als inhoud een set van procedures en functies voor het creëren en besturen van werkbalken, volgbalken en vooruitgang balken. De library moet geladen worden in uw programma met gebruik van onderstaande regel:

```
INSTALL @lib$+"WINLIB3"
```

De set procedures en functies zijn:

- FN_createfloatingtoolbar
- PROC_showfloatingtoolbar
- PROC_removefloatingtoolbar
- FN_custombuttons
- FN_createtrackbar
- PROC_showtrackbar
- FN_trackbarpos
- PROC_removetrackbar
- FN_createprogressbar
- PROC_showprogressbar
- PROC_stepprogressbar
- PROC_removeprogressbar

FN_createfloatingtoolbar(nbutts%,button%(),buttid%(),xpos%,ypos%,title$)

Deze functie maakt een zwevende werkbalk en geeft als resultaat de vensteringang (window handle). Het werkt op dezelfde manier als de FN_createtoolbar, behalve dat het drie extra parameters nodig heeft: de geïnitialiseerde horizontale en verticale coördinaat van de werkbalk (gerekend vanuit linksboven van het uitvoervenster van BASIC) en een string om als titel in de werkbalk weer te geven. In tegenstelling tot FN_createtoolbar wordt die werkbalk niet weergegeven totdat u PROC_showfloatingtoolbar aanroept; u kunt dan de werkbalk verwijderen en herstellen zonder hem elke keer te hoeven maken. U vermijdt daarmee het verspillen van het geheugen.

Normaal gesproken heeft een zwevende werkbalk een titelbar. Dit kan overgeslagen worden door een andere style te kiezen na de aanroep van **FN_createfloatingtoolbar**, zie als volgt:

```
ftb% = FN_createfloatingtoolbar(nbutts%,button%(),buttid%(),50,50,"")
WS_BORDER = &800000
ftb%!16 AND= NOT WS_BORDER
```

Merk echter op dat als u dit doet de gebruiker niet de werkbalk kan verplaatsen, zodat het nodig kan zijn om de positie meer zorgvuldig in te stellen.

PROC_showfloatingtoolbar(ftb%)

Deze procedure geeft een voor gecreëerde zwevende werkbalk weer dat één parameter vereist. De parameter bevat de teruggegeven waarde van **FN_createfloatingtoolbar**. Het volgende codesegment creëert en toont een zwevende werkbalk met as inhoud drie knoppen: een **knippen** knop, **kopiëren** knop en **plakken** knop:

```
nbutts% = 3
DIM button%(nbutts%-1), buttid%(nbutts%-1)
button%() = 0, 1, 2
buttid%() = 100, 101, 102
ftb% = FN_createfloatingtoolbar(nbutts%,button%(),buttid%(),50,\
\      50,"Floating")
PROC_showfloatingtoolbar(ftb%)
```

Zie FN_createtoolbar (bulletin nr. 3) voor meer details. Als u berichten wilt sturen naar de werkbalk (bijvoorbeeld om het uiterlijk van de knoppen te bepalen), kunt u de middelen gebruiken die beschreven staan onder FN_createtoolbar, maar onthoud wel dat **de teruggegeven waarde vanuit FN_createfloatingtoolbar een *pointer* is naar de handle van de werkbalk; niet de handle van zichzelf.** Om bijvoorbeeld een button uit te schakelen (grijs maken), doen we dit:

```
TB_SETSTATE = 1041
SYS "SendMessage", !ftb%, TB_SETSTATE, buttid%, 0
```

Let wel, dat u alleen berichten naar een zwevende werkbalk kunt verzenden nadat het is weergegeven met PROC_showfloatingtoolbar.

PROC_removefloatingtoolbar

Deze procedure verwijderd een zwevende werkbalk die daarvoor met FN_createfloatingtoolbar gecreëerd is. U zult altijd de werkbalk moeten verwijderen voordat uw programma wordt beëindigd of terug gaat naar de directe mode:

```
PROC_removefloatingtoolbar(ftb%)
```

Omdat FN_createfloatingtoolbar ruimte reserveert in de *heap*, is het essentieel dat u de werkbalk verwijderd voordat u een CLEAR, CHAIN of RUN statement uitvoert. Het is beter om het niet te doen, want het is zeer waarschijnlijk dat *BBC BASIC for Windows* kan crashen.

FN_custombuttons(htool%,bmpfile$,nbutt%,buttid%())

Deze functie lijkt sterk op FN_custombutton, maar hiermee is het mogelijk images van meerdere knoppen aan te passen vanuit een bitmap bestand. Vier parameters moeten genomen worden: de handle van de werkbalk (als voorbeeld van FN_createtoolbar), de naam van een Windows Bitmap bestand met als inhoud de benodigde knop images (van links naar rechts), het nummer van de knop images in het bestand en een array van *identifiers* van de knoppen die u wilt wijzigen.

De functie retourneert TRUE als de knop images met succes zijn gewijzigd, en anders FALSE (meest voorkomend door een gespecificeerd bestand kon niet gelezen worden of heeft een ongeldig formaat). U kunt zoveel knoppen aanpassen als u wilt door een bitmap te gebruiken met verschillende afmetingen.

Onthoud dat wanneer u een zwevende werkbalk gebruikt, **de teruggegeven waarde van FN_createfloatingtoolbar een *pointer* is naar de werkbalk handle; niet de handle van zichzelf.** Dus om alle knoppen in een zwevende werkbalk aan te passen:

```
ok% = FN_custombuttons(!ftb%, "birds.bmp", nbutts%, buttid%())
```

Wanneer u de werkbalk creëert, moet u het knop type van elke niet aangepaste knop instellen tot en met 15 (geen afbeelding). Zorg er ook voor dat de achtergrondkleur van uw plaatjes is R=192, G=192,

B=192 (&C0C0C0). Deze maatregelen zullen ervoor zorgen dat uw aangepaste afbeeldingen correct worden weergegeven.

FN_createtrackbar(hwnd%,xpos%,ypos%,width%,height%,style%)

Deze functie creëert een *volgbalk* (soms genoemd een *schuifregelaar*). Met een volgbalk kan de gebruiker een parameter bepalen door de schuifregelaar naar de gewenste positie te slepen. Er moeten zes parameters aanwezig zijn: de *window handle* van het ouder venster waar de volgbalk in staat (meestal **@hwnd%** tenzij u de volgbalk in een child venster wilt plaatsen), de horizontale en verticale coördinaat van de volgbalk (in pixels berekent vanaf de linker bovenhoek van de ouder vensterbalk), de breedte en hoogte van de volgbalk (in pixels) en de volgbalk *style*. Mogelijke waarden voor de style zijn als volgt:

Style	Naam	Effect
1	TBS_AUTOTICKS	Toon maatstreepjes
2	TBS_VERT	Verticale volgbalk
4	TBS_LEFT	Maatstreepjes naar boven of naar links (standaard is onder of rechts)
8	TBS_BOTH	Maatstreepjes aan beide zijden

De waarden kunnen gecombineerd worden.

PROC_showtrackbar(tb%,max%)

Deze procedure geeft een volgbalk weer die gecreëerd is met FN_createtrackbar. Het heeft twee parameters nodig: de teruggegeven waarde van FN_createtrackbar en de maximale waarde waarop de schuifregelaar verplaatst kan worden. Als de maatstreepjes getoond worden, dan is de totale waarde (inclusief beide einden) **max%+1**.

Voor het creëren van een horizontale volgbalk met maatstreepjes, is het geschikt om een waarde te kiezen tussen nul en tien:

```
tb% = FN_createtrackbar(@hwnd%, 100, 200, 20, 300, 1)
PROC_showtrackbar(tb%, 10)
```

FN_trackbarpos(tb%)

Deze functie retourneert de huidige positie van de volgbalk (tussen nul en de waarde van **max%** geleverd aan PROC_showtrackbar):

```
trackpos% = FN_trackbarpos(tb%)
```

Normaal wordt de volgbalk verplaatst door de gebruiker, maar als u wilt dat uw programma het verplaatst naar een specifieke positie, dan kunt u dat doen als volgt:

```
TBM_SETPOS = 1029
SYS "SendMessage", !tb%, TBM_SETPOS, 1, trackpos%
```

(vergeet niet het uitroepteken in **!tb%**).

PROC_removetrackbar(tb%)

Deze procedure verwijderd een volgbalk die eerder gecreëerd is met FN_createtrackbar. U moet altijd de volgbalk verwijderen voordat u uw programma sluit of terugkeert naar de directe modus:

```
PROC_removetrackbar(tb%)
```

Omdat FN_createtrackbar ruimte reserveert op de *heap*, is het essentieel dat u de volgbalk verwijderd voordat u een CLEAR, CHAIN of RUN statement uitvoert. Het kan dan lelijk gaan mislukken en zelfs uitkomen tot een *BBC BASIC for Windows* crash.

FN_createprogressbar(hwnd%,xpos%,ypos%,width%,height%,style%)

Deze functie maakt een voortgangsbalk die meestal gebruikt wordt om de gebruiker van de voortgang van een tijdrovende bewerking te informeren. Zes parameters moeten worden aangeleverd: *window handle* van het ouder venster waar de voortgangsbalk in staat (meestal **@hwnd%** tenzij u de voortgangsbalk in een child venster wilt plaatsen), de horizontale en verticale coördinaat van de voortgangsbalk (in pixels berekent vanaf de linker bovenhoek van de ouder vensterbalk), de breedte en hoogte van de voortgangsbalk (in pixels) en de voortgangsbalk *style*. Een style waarde nul selecteert een horizontale voortgangsbalk met afzonderlijke 'blokken'. Andere waarden zijn als volgt:

Style	Effect
1	Vooruitgang 'blokken' zijn aaneengesloten
4	De voortgangsbalk is verticaal

De waarden kunnen worden gecombineerd.

PROC_showprogressbar(pb%,max%)

Deze procedure geeft een voortgangsbalk weer gecreëerd met FN_createprogressbar. Het vereist twee parameters: de waarde die wordt geretourneerd door FN_createprogressbar en de maximale waarde die kan worden aangegeven. Bijvoorbeeld als de voortgang wordt weergegeven in procenten, kan **max%** gelijk aan 100 zijn.

Voor het maken en weergeven van een voortgangsbalk is een waarde tussen nul en tien geschikt:

```
pb% = FN_createprogressbar(@hwnd%, 100, 200, 20, 300, 0)
PROC_showprogressbar(pb%, 10)
```

PROC_stepprogressbar(pb%,step%)

Deze procedure voorziet de voortgangsbalk met een aantal stappen, gegeven bij **step%** (die een negatieve waarde kan hebben Als het eindresultaat groter is dan de waarde van **max%** (meegeleverd aan PROC_showprogressbar), of een negatief getal is, wordt die voortzet omgekeerd.). Om de voortgangsbalk met één stap te voorzien:

```
PROC_stepprogressbar(pb%, 1)
```

U kunt ook de voortgangsbalk tot een *absolute* waarde als volgt plaatsen:

```
PBM_SETPOS = 1026
SYS "SendMessage", !pb%, PBM_SETPOS, progress%, 0
```

PROC_removeprogressbar(pb%)

Deze procedure verwijderd een voortgangsbalk die eerder gecreëerd is met FN_createprogressbar. U moet altijd eerst de voortgangsbalk verwijderen voordat uw programma wordt beëindigd of terugkeert naar de directe modus:

```
PROC_removeprogressbar(pb%)
```

Omdat FN_createprogressbar ruimte reserveert op de *heap*, is het belangrijk dat u de voortgangsbalk verwijdert voordat u een CLEAR, CHAIN of RUN statement uitvoert. Het kan dan lelijk gaan mislukken en zelfs uitkomen tot een *BBC BASIC for Windows* crash.

BBC BASIC for Windows library help
Tekstvertaling van Engels naar Nederlands voorbehouden
Volgende library: Eigenschappenvensters en wizards

API's voor Liberty BASIC (2).

PTR
PTR wordt net zo gebruikt als een struct, maar het bestaat niet uit een memberlijst zoals een struct. Het bestaat uit een tekenreeks van karakters. Wanneer een string doorgegeven wordt in een API functie "AS PTR", wordt een 32 bit geheugenadres van de string doorgegeven, niet de string zelf. Tekst tussen aanhalingstekens mag worden gebruikt binnen een API aanroep als een PTR, maar het mag ook met een stringvariabele. Hier is een functie die het bijschrift wijzigt op de titelbalk van een venster. Argumenten omvatten de handle van het venster waarvan het bijschrift moet worden gewijzigd, en de nieuwe tekenreeks om hiervoor te gebruiken is een letterlijke tekenreeks die wordt doorgegeven als een PTR.

```
CallDll #user32, "SetWindowTextA",_
     winHandle as ulong,_
     "Nieuwe Bijschrift!" as ptr,_
     result as void
```

Het nieuwe bijschrift mag ook doorgegeven worden als een stringvariabele:

```
NewText$ = "Deze tekst was gewijzigd!"

CallDll #user32, "SetWindowTextA",_
     h as ulong,_          'handle van venster of control
     NewText$ as ptr,_     'nieuwe tekst string
     result as void        'geen resultaat
```

PTR stringresultaten
Net als sommige functies waarden geven aan struct members die ontvangen worden door de programmeur, kunnen sommige functies strings vullen die gebruikt worden door de programmeur. In het SetWindowText voorbeeld wordt de string niet gewijzigd. Het heeft de informatie nodig als inhoud in de string. Wijzigt de functie niet de string, dan zal Liberty BASIC een kopie met inhoud in de functie doorgeven.

Als de functie de inhoud van de string wijzigt, zal de string als een stringvariabele doorgegeven moeten worden die beëindigd is met een NULL karakter, dat een chr$(0) is. Is de null beëindiging gebruikt, dan zal de "AS PTR" instructie geen kopie van de string maken. Het vertelt de functie waar de locatie in het geheugen is van de string. Dit wordt het geheugenadres genoemd. Weet de functie het geheugenadres van de string, dan kan het de string wijzigen. Een nieuwe inhoud van de string kan dan worden gebruikt door de programmeur.

Een functie die het tegenovergestelde doet van bovenstaand voorbeeld is GetWindowTextA. Het retourneert het bijschrift van een venster. De tekst string moet eerst null zijn zodat het geheugenadres doorgegeven wordt aan de functie. Nadat de functie is aangeroepen, is de inhoud van het bijschrift in de PTR variabele.

```
Caption$ = space$(128) + chr$(0) : length = len(Caption$)

CallDll #user32, "GetWindowTextA",_
     winHandle as ulong, Caption$ as ptr,_
     length as long, result as long
```

Na de aanroep van deze functie, is het bijschrift, van welke de handle doorgegeven is in de aanroep, gevuld in de variabele Caption$. Voor de aanroep bevat de variabele alleen een string met een lege ruimte. Wanneer dit gebruikt wordt, kan een stringvariabele als een BUFFER worden beschouwd. Voordat de API aanroep plaatsvindt, zal niets op het scherm worden getoond wanneer het volgende commando wordt gegeven: `Print Caption$`

Als op de titelbalk van het venster staat: “Mijn Grote Programma”, dan zal na het aanroepen van het print commando `Print trim$(Caption$)`, het volgende staan:

```
Mijn Grote Programma
```

Merk op het gebruik van de TRIM$ functie. Zonder de functie zal de null beëindiging van de string verschijnen als een niet-afdrukbaar karakter (zwart rechthoekje) in het weergavevenster. Wanneer gebruik wordt gemaakt van een buffer, houd er dan rekening mee om het zo lang mogelijk te maken voor de opslag van de inhoud die geroutineerd zal worden. Als het niet lang genoeg is, dan kan alleen een deel van de informatie worden teruggegeven. Dit voorbeeld laat een buffer zien die te kort is:

```
Caption$ = space$(6) + chr$(0)
TextLength = len(Caption$)
CallDll #user32, "GetWindowTextA",_
     winHandle as ulong, Caption$ as ptr,_
     TextLength as long, result as long
Print trim$(Caption$)
```

...resulteert in de volgende weergave:

```
Mijn G
```

Zorg ervoor dat u variabelen elke keer opnieuw instelt die gebruikt worden als buffers, die ook als buffers zijn gebruikt, om er zeker van te zijn dat ze de juiste lengte hebben en geen resterende karakters bevatten van uit vorig gebruik:

```
Caption$ = space$(128) + chr$(0) : TextLength = len(Caption$)
```

WINSTRING()
Soms retourneert een API aanroep een pointer naar een string na de pointer doorgegeven is als een long waarde. De WINSTRING() functie accepteert die long waarde en geeft een string terug. De WINSTRING() functie is zeer geschikt wanneer de tekststring vanuit een STRUCT opgehaald moet worden die veranderd is door een functie. De volgende pseudo code demonstreert die methode.

```
Struct MyStruct, num as long
calldll #dll, "MyFunction", MyStruct as struct, result as long
changedValue = MyStruct.num.struct
```

Aanroepingen vertalen naar LB syntax

API's vertalen vanuit andere programmeertalen

API aanroepen die in algemene documenten staan, worden niet verwezen in Liberty BASIC formaat, het is dus handig om te weten hoe het vertaald moet worden naar Liberty BASIC syntax van C en Visual Basic syntax, omdat deze talen vaak in de documentatie gebruikt worden. Het uitleggen van C en Visual Basic in detail valt echter buiten het bestek van dit onderwerp in de Bulletin. Deze kleine steekproef moet dienen om te laten zien wat de overeenkomsten en verschillen zijn, en dat Liberty BASIC programmeurs een start hebben op het vertalen van API-aanroepen naar Liberty BASIC syntaxis.

Eerst een woord over types:

Een **byte** is een 8 bit integer zonder teken. Liberty BASIC heeft niet direct een tegenhanger. Wanneer het gebruikt wordt als een member van een struct, kunnen bytes worden doorgegeven als **char[n]**.
Een **integer** is een 32 bit waarde zonder teken. Converteer het naar **long** in Liberty BASIC syntax.
Een **word** is een 16 bit waarde zonder teken. Het bestaat uit twee **bytes** – een hoge byte en een lage byte. Converteer het naar **word** in Liberty BASIC syntax.
Een **uint** is een 32 bit waarde zonder teken. Converteer het naar **ulong** in Liberty BASIC syntax.
Een **long** is een 32 bit integer met teken. Converteer het naar **long** in Liberty BASIC syntax.
Een **ulong** is een 32 bit integer zonder teken. Converteer het naar **ulong** in Liberty BASIC syntax.
Een **handle (of hwnd)** is een 32 bit integer zonder teken. Converteer het naar **ulong** in Liberty BASIC syntax.
Een **dword** is een 32 bit integer zonder teken die bestaat uit twee words – een hoge word en een lage word. Converteer het naar **dword** of **ulong** in Liberty BASIC syntax.
Een **lpstr** is een 32 bit pointer naar een tekenreeks (karakterstring). Converteer naar **ptr** in Liberty BASIC syntax.
Een **far** aanwijzing, of een andere aanwijzing die start met **lp** is een 32 bit geheugenadres pointer. In Liberty BASIC kunnen een van deze twee toegang hebben met een **ptr** of een **struct**.
Een **struct** of **type** is vergelijkbaar naar een **lpstr** omdat het een 32 bit pointer is naar een geheugensegment adres. Gebruik in Liberty BASIC een **struct**.
Een **bool** is een 16 bit boolean waarde. Converteer het naar **boolean** in Liberty BASIC syntax. Een boolean waarde van 0 is FALSE, anders is het TRUE.

Visual Basic API's converteren

Het is gemakkelijk fouten te veroorzaken wanneer API aanroepingen worden gemaakt, dus maak gebruik van referentiematerialen om te verzekeren dat API functies correct worden gebruikt.

Hier is een generieke aanroep in Visual Basic syntax, vanuit het win32api.txt bestand die aanwezig is in verschillende plaatsen op internet:

```
Declare Function FunctionName Lib "DLLname" Alias "FunctionName" (ByVal
param1 as type, ByVal param2 as type) As type
```

In de generieke VB aanroep begint een Declare Function altijd met een aanroep, gevolgd door een naam van de functie en het woord Lib. De naam van de DLL staat tussen aanhalingstekens. Een Alias naam, als die gebruikt wordt, wordt weergegeven tussen aanhalingstekens. Als volgende zijn de parameters gegroepeerd tussen haakjes op volgorde, elk met een gespecificeerd type. Na het gesloten haakje wordt het resulteertype gegeven. Hier is een API aanroep om een rechthoek te tekenen in Visual Basic syntax:

```
Declare Function Rectangle Lib "gdi32" Alias "Rectangle" (ByVal hDC As
Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As
Long) As Long
```

Het is niet moeilijk om die aanroep naar Liberty BASIC syntax te converteren.

*** Declare Function – is het VB declaratie statement. In Visual Basic is het nodig om van te voren een functie te declareren voor later gebruik. In Liberty BASIC is dat niet vereist.

*** Rectangle – is de naam van de functie. In Liberty BASIC staat het tussen aanhalingstekens: “Rectangle”

*** Alias “FunctionName” – als de naam verschillend is van de functienaam, eerder getoond, gebruik die dan wanneer u het converteert naar Liberty BASIC syntax. Later meer over Alias.

*** ByVal hDC As Long – de eerste parameter die doorgegeven wordt in de aanroep. In dit geval is hDC de handle van een device context. Een LONG in Visual Basic is ook een LONG of ULONG in Liberty BASIC. Het voorvoegsel “h” geeft een handle aan, en handles moeten doorgegeven worden als ULONG. Het sleutelwoord ‘BYVAL’ zegt dat de waarde van de variabele gepasseerd wordt, niet het adres van de variabele zelf. Met andere woorden: de functie ontvangt een kopie van de variabele, niet het geheugenadres van de variabele. Liberty BASIC geeft altijd numerieke variabelen door bij waarde. Liberty BASIC syntax: hDC as ulong

*** ByVal X1 As Long – de linksboven x coördinaat van een rechthoek
*** ByVal Y1 As Long – de linksboven y coördinaat van een rechthoek
*** ByVal X2 As Long – de rechtsonder x coördinaat van een rechthoek
*** ByVal Y2 As Long – de rechtsonder y coördinaat van een rechthoek

- in Liberty BASIC: X1 as long, Y1 as long, X2 as long, Y2 as long

***) As Long – merk op dat alle parameters tussen haakjes staan, maar de laatste niet. De laatste is een speciale, omdat die de resulteerparameter identificeert. In Liberty BASIC syntax: result as long

C API’s converteren
Hier is dezelfde functie om een rechthoek te tekenen in C syntax:

```
BOOL Rectangle(
     HDC hdc,       // handle van device context
     int nLeftRect, // x coördinaat van linker bovenhoek van de rechthoek
     int nTopRect,  // y coördinaat van linker bovenhoek van de rechthoek
     int nRightRect,     // x coörd. van rechter onderhoek van de rechthoek
     int nBottomRect     // y coörd. van rechter onderhoek van de rechthoek
     );
```

In een C functieaanroep is de eerste parameter het resulteertype. De volgende is de naam van de functie. Na de functienaam zijn de parameters gegroepeerd op volgorde tussen haakjes, samen met hun aangegeven types.

*** BOOL – de eerste parameter getoond in C syntax is het resulteertype. Merk op dat, net als in VB, die parameter niet tussen haakjes te zien is, zoals wel bij de andere parameters het geval is. In Liberty BASIC: r as boolean

*** Rectangle – is de naam van de functie. De DLL die nodig is, is niet aanwezig in de aanroep, omdat de API functies getoond worden in gescheiden header bestanden in de C taal. In Liberty BASIC: CallDll #gdi32, “Rectangle”

*** HDC – de volgende parameter hDC is de handle van de Device Context die op die manier aan de functie doorgegeven wordt als een 32 bit waarde zonder teken, dat een ULONG is in Liberty BASIC. Merk op dat de parameters gegroepeerd zijn tussen haakjes, net zoals in VB syntax. In Liberty BASIC: hDC as ulong

*** int nLeftRect – de linksboven x coördinaat van een rechthoek
*** int nTopRect – de linksboven y coördinaat van een rechthoek
*** int nRightRect – de rechtsonder x coördinaat van een rechthoek
*** int nBottomRect – de rechtsonder y coördinaat van een rechthoek

- in Liberty BASIC syntax: X1 as long, Y1 as long, X2 as long, Y2 as long

Drie programmeertaalvergelijkingen *Rectangle*:

C syntax:
```
BOOL Rectangle(
     HDC hdc,       // handle van device context
     int nLeftRect, // x coördinaat van linker bovenhoek van de rechthoek
     int nTopRect,  // y coördinaat van linker bovenhoek van de rechthoek
     int nRightRect,     // x coörd. van rechter onderhoek van de rechthoek
     int nBottomRect     // y coörd. van rechter onderhoek van de rechthoek
     );
```

VB syntax:
```
Declare Function Rectangle Lib "gdi32" Alias "Rectangle" (ByVal hDC As
Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As
Long) As Long
```

Liberty BASIC syntax:
```
CallDll #gdi32, "Rectangle", hDC as ulong, X1 as long, Y1 as long,_
     X2 as long, Y2 as long, result as boolean
```

Gebrek aan overeenstemming

Soms is er gebrek aan overeenstemming tussen de documentatie voor C en voor Visual Basic. Voor de Rectangle functie, hier gedemonstreerd, heeft de C versie een resulteertype BOOL, terwijl de Visual Basic versie een resulteertype LONG heeft. Probeer het eerst vanuit de Visual Basic versie te converteren, verwijs dan, als er een probleem is, naar de C documentatie.

Alias
Het win32api.txt bestand voor Visual Basic laat soms een andere functienaam na het woord “Alias” zien in de documentatie. Als de functienaam anders is zorg er dan voor dat u de versie gebruikt getoond na “Alias”, omdat dit de naam is die Windows gebruikt voor de functie. Hier is een voorbeeld:

```
Declare Function GetWindowText Lib "user32" Alias "GetWindowTextA" (ByVal
hwnd As Long, ByVal lpString As String, ByVal cch As Long) As Long
```

De functie wordt eerst genoemd “GetWindowText”, maar het wordt gegeven met de alias “GetWindowTextA”. De “A” is belangrijk en moet in de functienaam verschijnen wanneer het genoemd wordt in Liberty BASIC. De “A” staat voor “Ansi” wat een karakterset inhoudt. De “A” is toegevoegd aan functienamen die tekststring parameters gebruiken. De C documentatie heeft niet altijd de eventuele functie-

naam, daarom is het een goed idee om een terug-check uit te voeren op het Visual Basic win32api.txt bestand voor de functie als u wilt converteren naar LB syntax. De “GetWindowTextA” functie in Liberty BASIC syntax:

```
CallDll #user32, "GetWindowTextA", hWnd as long, txt$ as ptr,_
     txtLen as long, r as long
```

Een side bij side vergelijking voor een rechthoek:

Liberty BASIC	**Visual Basic**	**C**
(#gdi32 wordt herkend door Liberty BASIC)	Lib “GDI32”	Gegeven in een apart header bestand
CallDll #gdi32, “Rectangle”	Declare Function Rectangle	Rectangle
HDC as ulong,_	(ByVal hDC As Long,	(HDC hDC,
X1 as long,_	ByVal X1 As Long,	int nLeftRect,
Y1 as long,_	ByVal Y1 As Long,	int nTopRect,
X2 as long,_	ByVal X2 As Long,	int nRightRect,
Y2 as long,_	ByVal Y2 As Long)	int nBottomRect)
Result as boolean	As Long	BOOL

Volgende Bulletin: De Windows API in Actie

Foutafhandelingen.

Hoe goed we ook een programma willen maken, nooit zullen we een programma 100 procent foutloos kunnen schrijven. Dit heeft niets te maken of we nou wel of geen ervaring hebben, want ook een professor kan fouten maken. Het ergste is dat fouten in een klein hoekje kunnen achterblijven als tandplak tussen smalle openingen waar je met je tandenborstel niet goed bij kunt komen.

Er bestaan een heleboel soorten fouten: grove fouten, formulefouten, compilerfouten en de gemeenste fouten zijn de schoonheidsfoutjes die altijd als laatste achterblijven. Schoonheidsfoutjes zijn ook het moeilijkst op te sporen. Daarom is het ook zo belangrijk om programma’s goed te structureren.

Programmafouten
De programmafouten kunnen we verdelen in grammatica- en spellingfouten, structurele fouten (deels door de compiler niet waar te nemen). De eerste twee zijn de makkelijkst op te sporen fouten, omdat er statements staan die onbekend zijn in de taal. In sommige programmeertalen neemt de compiler tijdens het typen van de code direct maatregelen. Een onbekend statement wordt meestal gezien als een variabele die niet gedeclareerd is. BASIC dialecten die variabelendeclaraties niet kennen, nemen dan ook geen maatregelen. Dit kan grote nadelige gevolgen hebben voor de uitvoer van de code. Als tweede, de structuren, bepalen de volgorde van de uitvoer van een programma. Als deze volgorde niet klopt dan kunnen er fouten ontstaan die alleen met de programmaloop te maken heeft. Soms kan de compiler het ontdekken wat er fout gaat, maar dat is niet altijd het geval. Een programma kan altijd wel werken, ook al is de programmaloop onjuist.

Uitvoerfouten
Als een programma werkt, dan kan men nog niet 100 procent zeggen dat het programma ook echt goed is. Eerst moet een programma getest worden door allerlei soorten invoer te gebruiken. Wat gebeurt er als er een letter wordt ingetoetst, terwijl een getal nodig is, en hoe reageert het programma als per ongeluk gedeeld wordt door nul? Dit soort fouten gebeurt alleen tijdens de uitvoer van een programma. Of er per ongeluk door nul wordt gedeeld, kan de compiler nooit weten.

Schoonheidsfoutjes
De kleinste foutjes zitten in een klein hoekje. Wat zijn het eigenlijk voor foutjes? Het onderstaande is wel op te noemen:

- Zwerfvariabelen
- Niet gebruikte procedures en functies
- Geen invoercontrole of de controle klopt niet
- Gehele getallen gebruiken in plaats van getallen met decimalen of andersom
- Verkeerde doorvoer van de gegevens (procedure-, functieparameters; dialoogvensters; gegevensbestanden inlezen, wegschrijven, enzovoort)

Fouten afhandelen
De schoonheidsfoutjes zijn moeilijk te vinden, daarom is het belangrijk voor een bepaalde uitvoer actie te ondernemen. Dat kan op verschillende manieren, actie van de programmeur of de actie automatisch laten afhandelen. De tweede manier is niet altijd zinvol, zoals onderstaande code laat zien:

```
     ...
     Getal = Val(txtGetal.Text)
     On Error Goto DelenDoorNul
     N = M / Getal
     On Error Goto 0
Exit Sub
     DelenDoorNul:
     MsgBox "Fout! Delen door nul kan niet!"
End Sub
```

Hoewel de code goed werkt en de invoer prima afgehandeld wordt, is dit toch af te raden. Wie kan zeggen dat de gebruiker werkelijk een nul invoert? Niemand. Er kan dus worden gezegd: “Fout! Delen door nul kan niet!”, terwijl de gebruiker misschien wel een letter heeft ingevoerd.

Dit moet dus afgehandeld worden op de eerste manier, actie van de programmeur. De programmeur moet ervoor kunnen zorgen dat er werkelijk een nul is ingetoetst of een letter. Twee foutafhandelingen dus:

```
...
Getal = Val(txtGetal.Text)
If Getal <> 0 Then
     N = M / Getal
Else
     If Trim$(txtGetal.Text) <> "0" Then
          MsgBox "Fout! Verkeerde invoer, alleen getallen!"
     Else
          MsgBox "Fout! Delen door nul kan niet!"
     End If
End If
...
```

Nu is het veel gemakkelijker te begrijpen en is de foutafhandeling goed te volgen, ook al kost het meer coderegels.

In Visual Basic .NET werkt het nu nog beter. Er bestaan functies, zoals **FileExists()**, die een boolean resulteert of het bestand wel of niet bestaat. We hoeven steeds minder fouten zelf af te handelen. De meeste controls doen dit al voor ons. Maar het kan geen kwaad extra controle, als bovenstaande code doet, in het programma op te nemen. U kunt zelfs eigen functies schrijven die bepaalde fouten afhandelen.

In het stukje code bovenaan is ook te zien dat ik een controle in een controle uitvoer. Met de gegevensinvoer kunnen er twee kanten worden opgegaan: getallen of tekst. Voordat ik het laat weten of de hele invoer fout is of dat er een nul ingevoerd is, controleer ik eerst of ik de invoer kan converteren naar een numerieke waarde. De functie **Val** heeft echter een nadeel; de functie resulteert een nul als de invoer niet numeriek is, daarom controleer ik genest, na de **Else,** nogmaals de invoer met het tekstvak zelf, zonder de Val functie.

Strenger maken
Door dit steeds gedetailleerder te doen, wordt de uitvoer van het programma strenger. Er wordt steeds meer op verschillende soorten invoer gelet. U kunt zelfs controleren of het getal decimalen heeft of niet!

Verkeerd gebruik van de functie Abs()
De functie Abs() van Absolute, zet negatieve waarden om in positieve waarden. Dat kan best nuttig zijn, maar als u de gebruiker wilt attenderen om wat betreft de invoer, dan heeft de functie een nadeel. Maar ook dit is op te lossen:

```
If Getal < 0 Then
      If MsgBox("Getal is negatief, mag deze positief worden gemaakt?", _
          vbQuestion Or vbYesNo, "Vraag") = vbYes Then
          Getal = Abs(Getal)
      End If
End If
```

Deze handelingen zullen niet met een *On Error Goto* of een *Trap … Resume* of een *Try … Exception* goed afgehandeld kunnen worden, we zouden dan, net als bij *If … Then … Else … End If*, meerdere code moeten nesten om ervoor te zorgen dat we uit één invoer of gelezen bestand meerdere berichten kunnen weergeven, want er kan gewoon meer gebeuren dan we met één handeling in orde zouden kunnen maken.

Globale variabelen
In veel BASIC dialecten hoeven de variabelen niet gedeclareerd te worden. In grote projecten kan het een ramp veroorzaken als hier niet goed mee wordt omgegaan. Globale variabelen is als rondzwerven in een dicht woud en nooit meer de rand van het bos kunnen terugvinden. Zwerfvariabelen kunnen een grote vuile code achterlaten. Het is oud papier even van het bureau halen en wegleggen in een donkere hoek.
De enige oplossing is om alle stukjes code die in subroutines en functies staan samen te laten werken, zoals we ook emailberichten naar elkaar kunnen doorsturen met dezelfde teksten en bijlagen, zonder dat we steeds weer alles opnieuw op te moeten zoeken en erin moeten zetten.

Beginners vinden het gebruik van parameters en argumenten in procedures en functies erg lastig en denken een makkelijkere weg te kunnen vinden door gewoon alle codeblokken met dezelfde variabelen te laten werken die in een module globaal worden gedeclareerd, of zelfs helemaal niet. Men vraagt zich dan ook af, waarom al die moeite? Als we terugkijken naar het codevoorbeeld, dan kunnen we de

variabele Getal op drie manieren beschouwen: globaal, lokaal of als argument, en evenzo de andere twee variabelen, N en M. Het zijn geen duidelijke namen, maar dat was ook even niet van belang.

Lokale variabelen
Lokale variabelen kunnen we ook beschouwen als *tijdelijke* variabelen. Ze worden gedeclareerd in codeblokken om delen van lange formules uit te kunnen voeren, zodat de te resulteren waarde gemakkelijk teruggegeven kan worden door de functie. In elk code blok mag dezelfde lokale variabele naam worden gebruikt, dit in tegenstelling bij globale variabelen die voor een hele programmawijziging kunnen zorgen als er een waarde wordt veranderd. Dit is dan ook gelijk het antwoord: Waarom al die moeite? Maar we hebben nog een derde soort variabele, waar het antwoord op te vinden is.

Argumenten
Een argument is als een doorgestuurd bericht en verwerkt worden. Degene die het bericht doorstuurt is de *parameter*. Meestal worden deze twee doorelkaar gehaald en heeft men het over: “Ik gebruik optionele argumenten bij de procedure aanroep.”, maar met de aanroep van een procedure of functie worden altijd de parameters gegeven, nooit de argumenten.

Is een argument lokaal? Het antwoord is Ja, maar het kan ook Nee zijn. Een argument kan op twee manieren in de declaratie gedeclareerd worden: bij waarde of bij referentie. Is het argument ‘bij referentie’ (ByRef), dan kunnen er fouten worden gemaakt als de parameter bepaalde waarden probeert door te geven. De waarde mag bijvoorbeeld geen expressie zijn, want zal de waarde van het argument veranderd worden, dan veranderd de parameter ook en dat kan alleen als een parameter als variabele wordt gebruikt.

De techniek in programmeren
Het is bijzonder belangrijk een goede techniek toe te passen waar fouten zo min mogelijk in voorkomen. Handel fouten goed af. Gebruik controlestructuren om fouten af te handelen. Lukt dat niet, handel dan de fouten af met de foutafhandeling statements. Er zijn zoveel soorten manieren om fouten tegen te gaan, maar iedere programmeur heeft zijn eigen techniek om zijn programma zo goed mogelijk te laten werken.

Strings gebruiken met de statements en functies.

Laten we eens gedetailleerder werken met tekst. De strings gebruiken we op een hele andere manier dan de getallen. We gaan daarom eens bekijken wat we met tekst kunnen doen in BASIC.

In de vorige Bulletin zag u voorbeelden hoe programmeertaal C met tekenreeksen omgaat. Die manier lijkt een stuk lastiger dan wanneer we gewoon een hele tekst toekennen aan een stringvariabele.

Naast de standaard stringstatements en stringfuncties bestaan er in Visual Basic .NET stringmethoden. Elke variabele en zelfs een expressie tussen haakjes worden herkend als objecten waar de stringprocedures en stringfuncties als methoden aangeroepen kunnen worden.

De lengte bepalen
Alle BASIC dialecten kennen de LEN functie, waarmee we de lengte van een string kunnen bepalen. De lengte wordt bepaald uit het aantal tekens, inclusief de tekens die onzichtbaar zijn, zoals de spaties.

```
L = LEN(A$)
L = LEN("Hoera")
L = Naam.Length()
L = ("Hoera").Length()
```

De laatste twee regels werken alleen in de Visual Basic .NET versies. In andere BASIC dialecten zijn de variabelen en expressies geen objecten. Dankzij het .NET Framework library erven alle objecten de methoden van het basis object. Er bestaat dus maar één Length() functie die in het basis object gemaakt is.

Een probleem oplossen; hoe rijgen we tekstdelen aan elkaar en hoe lezen we het er weer uit?
Eén van de grootste problemen is het bewerken van een string en het uitlezen van een string. Niet door het in één keer uit te lezen, maar het in delen uit te lezen. Dit is een probleem, omdat niet altijd een goede manier gevonden wordt, hoe we tekst aan elkaar kunnen zetten en er weer uit kunnen halen. De gegeven LEN functie is al een deel van de oplossing, want die zullen we moeten gebruiken om de lengte van elk tekstdeel te kunnen bepalen.

De aanpak
Stel dat het gaat om een monteur die op afspraak met de gekozen dagdelen naar een klant moet. De afspraak is dat de monteur drie dagdelen nodig heeft om bij een klant zijn werkzaamheden uit te kunnen voeren.
We hebben snel de neiging om een array te declareren en in elk element een dagdeel in te bewaren.

```
Dagdeel$(0) = "Begin Ochtend"
Dagdeel$(1) = "Ochtend 1"
Dagdeel$(2) = "Ochtend 2"
```

Als we het aantal dagdelen weten, dan kunnen we gewoon een array declareren met die aantal.

Begin Ochtend, Ochtend 1, Ochtend 2, Eind Ochtend, Begin Middag, Middag 1, Middag 2, Middag 3, Eind Middag

Het aantal is dus negen, dus:

```
DIM Dagdeel$(9)     ' Of: Dim DagDeel(9) As String
```

Maar is dit wel de goede aanpak voor het bewaren en uitlezen van de dagdelen? Stel dat er een afspraak gemaakt wordt en de monteur heeft maar één dagdeel nodig. Dan zitten we met één gevuld element in een array van negen elementen. Er is een betere oplossing waar zelfs een array niet voor nodig is, maar wel wat meer code vergt om de oplossing te laten werken.

De aanpak is om gewoon de dagdelen op te slaan in één variabele, en wel door bovenstaande dagdelen gescheiden door een puntkomma, zonder spaties, op te slaan. Niet alle dagdelen, maar alleen de dagdelen die voor een monteur nodig zijn. We kunnen dit doen met een functie die wel in de meeste BASIC dialecten bestaat.

De functie INSTR
Met de functie INSTR kunnen we zoeken naar tekstdelen en bepalen op welke plaats het deel aanwezig is. We zouden dus kunnen zoeken naar elk dagdeel in een string, maar dat is echter niet de bedoeling. We willen niet naar de dagdelen zoeken, maar ze uit de string halen. We moeten dus vanaf het begin van de string tot het einde van de string lopen en ondertussen de dagdelen eruit halen. We lopen echter niet van teken tot teken, maar we gaan zoeken naar het scheidingsteken.

Een pseudovoorbeeld zou zijn:
p = INSTR(string, “;”)

Maar dit is niet alles. We moeten controleren of er nog puntkomma’s zijn, daarna moeten we de string doorhakken met p – 1. Eén er van aftrekken is belangrijk, om niet de puntkomma mee te nemen.

Daarna doet de programmeur iets met het linkerdeel. (Hier niet van belang.) Het rechterdeel zal de nieuwe string zijn waar opnieuw de lus zal worden herhaald en opnieuw gezocht zal worden naar een volgende puntkomma, enzovoort, totdat er geen puntkomma's meer zijn en p gelijk zal zijn aan nul.

Onderstaande lus laat een uitgebreid voorbeeld zien.

```
Gevonden = FALSE
DO
     p = INSTR(Dagdelen$, ";")
     IF p > 0 THEN
          Dagdeel$ = LEFT$(Dagdelen$, p - 1)
          ' Doe hier iets met Dagdeel$
          Dagdelen$ = RIGHT$(Dagdelen$, LEN(Dagdelen$) - LEN(Dagdeel$))
          ' Ook mogelijk is:
          ' Dagdelen$ = MID$(Dagdelen$, LEN(Dagdeel$) + 2)
          Gevonden = TRUE
     ELSE
          Dagdeel$ = Dagdelen$
          ' Doe hier iets met Dagdeel$
          Gevonden = FALSE
     END IF
LOOP UNTIL NOT Gevonden
```

Let er wel op dat Dagdelen$ uit elkaar wordt gehaald. Zorg er dus voor dat Dagdelen$ een lokale variabele is die de dagdelen meekrijgt via een argument.

In plaats van de Dagdelen$ te overschrijven, zouden we ook de plaats kunnen onthouden om telkens naar de volgende puntkomma te zoeken. De lange RIGHT$ functie, of de MID$ functie, is dan niet meer nodig.

```
Gevonden = FALSE
p = 1
DO
     p1 = INSTR(p, Dagdelen$, ";")
     IF p1 > 0 THEN
          Dagdeel$ = MID$(Dagdelen$, p, p1 - 1)
          p = p1 + 1
          Gevonden = TRUE
     ELSE
          Dagdeel$ = MID$(Dagdelen$, p)
          Gevonden = FALSE
     END IF
LOOP UNTIL NOT Gevonden
```

Probeer het eens uit. Met de stringfuncties zijn veel tekstmogelijkheden te gebruiken. Zo kunt u zien dat het tweede voorbeeld een stuk inkort door de string juist niet door te hakken, maar van plaats te veranderen. De MID$ functie is dan zeer handig om delen uit een string te kunnen halen.

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 19:00 uur tot 21:00 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	ma. t/m zo.	19-22	06 30896598	m.a.kurvers@live.nl
Basic algemeen, zoals VBA Office	Marco Kurvers	ma. t/m zo.	19-22	06 30896598	m.a.kurvers@live.nl
Web Design, met XHTML en CSS