Basic Bulletin

21ste jaargang oktober 2014 Nummer 3

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

BBC BASIC gebruikt API's, zoals u ziet over de procedures en functies die in de library's staan. In Liberty BASIC wordt er ook gebruik gemaakt van API's. Hoe het in Liberty BASIC werkt, ziet u hier.

Hoe belangrijk is het te werken met expressies en tekenreeksen. De twee laatste hoofdstukken laten zien hoe belangrijk het kan zijn. Een goed idee om in nummer 4 daarmee verder te gaan.

Marco Kurvers

BBC BASIC for Windows – De library's (2).

Werkbalken en statusbalken.

In de vorige Bulletin was er de introductie over de library's en kwam daarna de eerste library aan bod: arrays en matrices.

De library die nu komt bevat mogelijkheden om werkbalken en statusbalken te gebruiken. De WINLIB bibliotheek (genoemd als de library) bevat een set van procedures en functies voor het maken en beheren van werkbalken en statusbalken. Hiermee kunt u uw BBC BASIC for Windows programma's meer laten lijken op echte Windows™ programma's, met een goede grafische User Interface.

De bibliotheek moet worden geladen vanuit uw programma met behulp van de opdracht:

```
INSTALL @lib$+"WINLIB"
```

De set procedures en functies zijn:

- FN_createstatusbar
- FN_createstatusbarex
- PROC_removestatusbar
- FN_createtoolbar
- FN_createtoolbarex
- PROC_removetoolbar
- FN_custombutton
- PROC_addtooltips

De WINLIBU bibliotheek bevat een identieke set van functies, behalve dat die een tekenreeks ontvangen in Unicode (UTF-8)-indeling, waardoor niet-ANSI (bijvoorbeeld de buitenlandse taal) tekens kunnen worden opgenomen.

FN_createstatusbar(text$)

Deze functie maakt een statusbalk en geeft als resultaat de vensteringang (window handle). Een parameter van één tekenreeks moet worden verstrekt, waarin de tekst (indien aanwezig) standaard moet worden weergegeven in de statusbalk. De geretourneerde waarde kan worden gebruikt voor het manipuleren van de statusbalk met behulp van Windows ™ API-functies (zie hieronder).

Het volgende programma segment maakt een statusbalk met de tekenreeks "Druk op F1 voor Help":

```
hstat% = FN_createstatusbar("Druk op F1 voor Help")
```

Onthoud dat het BASIC uitvoer venster niet automatisch smaller wordt om de statusbalk tegemoet te komen. U zult gebruik moeten maken van de opdrachten VDU 24 en/of VDU 28 om de afbeeldingen en/of de tekst vensters aan te passen. Raadpleeg de help van BBC BASIC voor meer informatie over die twee opdrachten. De opdrachten zullen we regelmatig in de bulletins tegenkomen.

Als u wilt dat de statusbalk zich automatisch van formaat wijzigt wanneer het formaat van het hoofdvenster wordt gewijzigd (bijvoorbeeld als de gebruiker een hoek versleept), moet u een bericht doorgeven dat de grootte wijzigt. Onderstaande regel doet dat:

```
ON MOVE SYS "PostMessage",hstat%,@msg%,@wparam%,@lparam% : RETURN
```

Door PROC_removestatusbar aan te roepen voordat het programma wordt afgesloten of terugkeert naar de directe modus, moet u de statusbalk verwijderen. Het is een goede gewoonte de fouten af te vangen (met behulp van ON ERROR) en de statusbalk te verwijderen als een onverwachte fout optreedt.

Zodra u een fundamentele statusbalk hebt gemaakt, kunt u Windows ™ API-functies verdelen in een aantal onderdelen en andere tekst schrijven in elk deel. Het volgende programma segment wordt geïllustreerd hoe u de statusbalk opsplitst in drie delen; hstat % is de resultaatwaarde van de functie FN_createstatusbar.

```
SB_SETTEXT = 1025
SB_SETPARTS = 1028
nparts% = 3
DIM edges%(nparts%-1)
FOR N% = 0 TO nparts%-1
  READ edges%(N%)
NEXT N%
SYS "SendMessage", hstat%, SB_SETPARTS, nparts%, ^edges%(0)
FOR N% = 0 TO nparts%-1
  READ text$
  SYS "SendMessage", hstat%, SB_SETTEXT, N%, CHR$9+text$
NEXT N%
DATA 440, 540, 640
DATA Part 1, Part 2, Part 3
```

De eerste DATA instructie bepaalt de breedte van elk onderdeel (part), door de positie van de rechterkant (in pixels) op te geven. De tweede DATA instructie bepaalt de tekst die in elk deel verschijnen zal; het teken van CHR$ 9 zorgt ervoor dat de tekst centraal wordt weergegeven.

FN_createstatusbarex(text$, style%)

Deze functie is vergelijkbaar met FN_createstatusbar behalve dat het een extra stijl% parameter heeft. Deze parameter kan worden ingesteld op SBARS_SIZEGRIP (256) dat tot een 'grootte grip' aan het extreem rechtse eind van de statusbalk zal worden weergegeven. Dit is geschikt wanneer het bovenliggende venster kan worden aangepast door de gebruiker (door een zijde of een hoek te slepen).

PROC_removestatusbar

Deze procedure verwijdert een statusbalk die eerder gecreëerd is met FN_createstatusbar of FN_createstatusbarex. U moet altijd de statusbalk verwijderen voordat het programma beëindigd wordt of teruggekeerd wordt naar de directe modus.

FN_createtoolbar(nbutts%,button%(),buttid%())

Deze functie creëert een werkbalk en resulteert de window handle. Drie parameters dienen te worden verstrekt: het aantal knoppen in de werkbalk, een integer array van knoptypes en een integer array van knop-id's. De geretourneerde waarde kan worden gebruikt voor het manipuleren van de werkbalk met behulp van Windows ™ API-functies (zie hieronder). Om een zwevende werkbalk te creëren heeft u de FN_createfloatingtoolbar functie in de WINLIB3 bibliotheek nodig.

In het volgende programma segment wordt een werkbalk met drie knoppen gemaakt: een knop knippen, een knop kopiëren en een knop plakken:

```
nbutts% = 3
DIM button%(nbutts%-1), buttid%(nbutts%-1)
```

```
button%() = 0, 1, 2
buttid%() = 100, 101, 102
htool% = FN_createtoolbar(nbutts%,button%(),buttid%())
```

De button% array bevat de knop *types* en de buttid% array bevat de knop *identifiers* (die gebruikt worden om de knoppen te identificeren en te bepalen op welke knop is geklikt). Is de knop *identifier* nul dan betekent dit een scheidingsteken (separator) in plaats van een knop.

In versie 1.7 en later van de WINLIB-bibliotheek kunt u een negatieve id-waarde opgeven. Hiermee maakt u een auto wisselknop die de knop 'ingedrukt' of 'uitgedrukt' bepaalt wanneer u erop klikt. In dit geval is de waarde van de werkelijke id toegewezen aan de knop minus de waarde die is opgegeven in de matrix van buttid%.

De aanwezige soorten knoppen zijn als volgt:

Waarde	**Knop**
0	Knippen
1	Kopiëren
2	Plakken
3	Ongedaan maken
4	Terugzetten
5	Verwijderen
6	Nieuw
7	Openen
8	Opslaan
9	Zoeken (zoek)
10	Afdrukvoorbeeld
11	Help
12	Zoek (zoeken)
13	Vervangen
14	Afdrukken
15	geen afbeelding

Onthoud dat het BASIC's uitvoervenster niet automatisch kleiner gemaakt is voor de werkbalk. U moet de VDU 24 en/of VDU 28 opdrachten gebruiken om de grootte van de afbeeldingen en/of tekst windows aan te passen.

Als u wilt dat de werkbalk zelf automatisch het formaat wijzigt wanneer het formaat van het hoofdvenster wordt gewijzigd (bijvoorbeeld als de gebruiker een hoek versleept), moet u het bericht met de grootte-verandering doorgeven. Hieronder ziet u de regel die dat als volgt doet:

```
ON MOVE SYS "PostMessage",htool%,@msg%,@wparam%,@lparam% : RETURN
```

U moet de werkbalk verwijderen door PROC_removetoolbar aan te roepen voordat het programma wordt afgesloten of terugkeert naar de directe modus. Het is een goede gewoonte om fouten te onderscheppen (met behulp van ON ERROR) en de werkbalk te verwijderen als een onverwachte fout optreedt.

Zodra u een werkbalk hebt gemaakt, dan zal het klikken op de knoppen voor berichten zorgen die verzonden worden naar uw programma zodat ze gedetecteerd kunnen worden met ON SYS, op dezelfde manier als met een selectie van een menubalk. De @wparam% waarde zal de knop identifier bevatten die gekozen wordt. Als alternatief kan het proces worden geautomatiseerd om het gebruik van ON SYS in uw code te voorkomen door de id toe te wijzen met behulp van FN_setproc in WINLIB5.

U kunt Windows ™ API-functies gebruiken om de vormgeving van de knoppen te bepalen. Het volgende programma segment wordt geïllustreerd hoe u een knop kunt uitschakelen (grijs); htool% is de resultaatwaarde van de functie FN_createtoolbar:

```
TB_SETSTATE = 1041
state% = 0
SYS "SendMessage", htool%, TB_SETSTATE, buttid%, state%
```

Enkele van de mogelijke knoptoestanden zijn:

0	Disabled
4	Enabled
6	Pressed
8	Hidden

U kunt de huidige status van een knop als volgt ontdekken:

```
TB_GETSTATE = 1042
SYS "SendMessage", htool%, TB_GETSTATE, buttid%, 0 TO state%
```

In het geval van een **auto wissel** knop die aan de ´ingedrukte´ staat is geweest, zal de resultaatwaarde één hoger zijn dan werd getoond. Zo is 5 dat deze was gewisseld naar de ´ingedrukte´ status en 7 dat deze werd ingedrukt door de gebruiker.

FN_createtoolbarex(nbutts%,button%(),buttid%(),resid%,style%)

Deze functie werkt op dezelfde manier als FN_createtoolbar, behalve dat het extra **resid%** en **style%** parameters bevat. Parameter **resid%** moet worden ingesteld op IDB_STD_SMALL_COLOR(0) voor kleine werkbalkknoppen of IDB_STD_LARGE_COLOR(1) voor grote werkbalkknoppen. Parameter **style%** kan worden ingesteld op één of meer van de werkbalkstijlen, zoals TBSTYLE_WRAPABLE(512) om de werkbalk terug te laten lopen op meerdere lijnen als de breedte ervan onvoldoende is.

PROC_removetoolbar

Deze procedure verwijderd een werkbalk die eerder was gecreëerd met FN_createtoolbar of FN_createtoolbarex. U moet altijd de werkbalk verwijderen voordat uw programma beëindigd wordt of het terugkeert naar de directe mode.

FN_custombutton(htool%,bmpfile$,buttid%)

Met deze functie kunt u een afbeelding van de knop aanpassen aan uw specifieke applicatie. Drie parameters dienen te worden verstrekt: de handle van de werkbalk (als resultaat FN_createtoolbar), de naam van een Windows bitmap bestand met de vereiste knopafbeelding (deze wordt automatisch aangepast aan de knop) en de *identifier* van de knop die u wilt wijzigen.

De functie resulteert TRUE als het knopimage succesvol is gewijzigd, en FALSE als waarschijnlijk het opgegeven bestand niet kon worden gelezen of een ongeldige indeling heeft. U kunt zoveel knoppen met verschillende waarden van **buttid%** aanpassen met deze functie. Als alternatief kunt u meerdere knoppen uit een enkel bitmapbestand met de FN_custombuttons functie gebruiken die te vinden is in de WINLIB3 bibliotheek.

Het volgende programmasegment illustreert hoe u een afbeelding van de knop aanpast:

```
ok% = FN_custombutton(htool%, "\pictures\bmp\owl.bmp", 102)
```

Wanneer in eerste instantie de werkbalk gecreëerd wordt, stel dan het knop ‘type’ van elke aangepaste knop op 15 (geen image). Zorg er ook voor dat de achtergrondkleur van uw bitmapbestand is R = 192, G = 192, B = 192 (&C0C0C0). Deze maatregelen zullen ervoor zorgen dat uw aangepaste afbeeldingen correct worden weergegeven.

PROC_addtooltips(htool%,nbutts%,buttip$(),buttid%())

Met deze procedure kunt u knopinfo toevoegen aan uw werkbalk. Knopinfo is een tekenreeks die wordt weergegeven in een geel vakje wanneer u met de muis over de knop op de werkbalk voor een korte tijd blijft hangen. Knopinfo toevoegen aan uw werkbalk biedt uw programma een zelfs meer authentiek Windows ™-interface.

Vier parameters dienen te worden verstrekt: de handle van de werkbalk (als resultaat FN_createtoolbar), het aantal knoppen waarvoor u tips wilt bieden, een matrix van tekenreeksen die de tips worden weergegeven en een geheel getal matrix met id-waarden voor de betreffende knoppen.

Het volgende programmasegment illustreert hoe u knopinfo toevoegt aan de werkbalk, gecreëerd in het vorige voorbeeld:

```
nbutts% = 3
DIM buttip$(nbutts%-1), buttid%(nbutts%-1)
buttip$() = "Cut", "Copy", "Paste"
buttid%() = 100, 101, 102
PROC_addtooltips(htool%,nbutts%,buttip$(),buttid%())
```

De matrix van **buttid%** mag vaak hetzelfde zijn als die doorgegeven aan FN_createtoolbar.

Dialoogvensters

De **WINLIB2** bibliotheek bevat een set van procedures en functies voor het maken en beheren van aangepaste dialoogvensters. Hiermee kunt u met de geprefereerde Windows ™-methoden de invoer van de gebruiker opvragen.

De bibliotheek moet worden geladen vanuit uw programma met behulp van de opdracht:

```
INSTALL @lib$+"WINLIB2"
```

De procedures en functies zijn:

- FN_newdialog
- PROC_pushbutton
- PROC_checkbox
- PROC_radiobutton
- PROC_groupbox
- PROC_editbox
- PROC_static
- PROC_listbox
- PROC_combobox
- PROC_dlgitem
- PROC_dlgctrl
- PROC_showdialog
- PROC_closedialog

De **WINLIB2B** bibliotheek bevat een identieke set van functies die worden doorgestuurd naar uw programma, zonder berichten van het **WM_HELP** en **WM_NOTIFY** die al gegenereerd worden door het dialoogvenster besturingselementen. Deze berichten kunnen worden onderschept met behulp van *** SYS 1** om uw programma aan te bieden, bijvoorbeeld *contextafhankelijke help* voor uw dialoog box.

WINLIB2B onderdrukt ook het automatisch sluiten in het dialoogvenster wanneer u op de knop Sluiten klikt, of op Escape drukt.

De **WINLIB2U** bibliotheek bevat een identieke set van functies van WINLIB2B, behalve dat die een tekenreeks ontvangen in Unicode (UTF-8)-indeling, waardoor niet-ANSI (bijvoorbeeld de buitenlandse taal) tekens kunnen worden opgenomen.

FN_newdialog

Voordat u een aangepast dialoogvenster kunt gebruiken, moet u eerst de positie, grootte en titel definiëren. Dit moet slechts eenmaal worden gedaan, meestal in een initialisatie routine voor elke dialoogvenster dat uw programma bevat (het dialoogvenster kan vervolgens worden weergegeven zo vaak als u nodig hebt):

```
dlg% = FN_newdialog(title$,x%,y%,cx%,cy%,font%,size%)
```

De tekenreeks **title$** is de titel van het dialoogvenster dat wordt weergegeven in de titelbalk. De waarden **x%** en **y%** is de eerste positie van het dialoogvenster, met betrekking tot de linker bovenhoek van uw programmavenster. De waarden **cx%** en **cy%** zijn respectievelijk de breedte en hoogte van het dialoogvenster.

De waarde **font%** is de grootte, in punten, van alle tekenreeksen in het dialoogvenster, en bepaalt ook de grootte van de eenheden die worden gebruikt om van de positie en grootte van het vak op te geven. Daarom, als u de grootte van de tekst wijzigt, worden alle andere dimensies van het dialoogvenster geschaald om aan te passen. Een gemeenschappelijke waarde die u moet gebruiken is 8.

De waarde **size%** is het aantal geheugenbytes, dat nodig is voor het dialoogvenster *sjabloon*. Als een vuistregel kan dit worden ingesteld op ongeveer het aantal afzonderlijke items in het dialoogvenster vermenigvuldigd met 50. Als de waarde te klein is ontvangt u het foutbericht **No room for dialogue template** wanneer de items zijn gemaakt. Het kan geen kwaad, afgezien van verspild geheugen, wanneer het te groot is.

De geretourneerde waarde (**dlg%** in dit geval) identificeert het dialoogvenster, en moet worden opgeslagen voor gebruik in de volgende procedures.

Een dialoogvenster heeft gewoonlijk een titelbalk. Dit kan worden als volgt worden opgeheven door verandering van de stijl na de aanroep van **FN_newdialog**:

```
dlg% = FN_newdialog("",x%,y%,cx%,cy%,font%,size%)
WS_BORDER = &800000
dlg%!16 AND= NOT WS_BORDER
```

Merk echter op dat als u dit doet de gebruiker het dialoogvenster niet kan verplaatsen, zodat het nodig kan zijn de positie zorgvuldiger in te stellen.

Zorg ervoor dat u alleen **FN_newdialog** eenmalig voor elk dialoogvenster aanroept; als u een dialoogvenster wilt re-openen dat al geopend is, roept u simpel **PROC_showdialog** aan. Bij voorkeur plaatst u alle **FN_newdialog** aanroepen in een initialisatieroutine die slechts één keer wordt uitgevoerd.

PROC_pushbutton

Zodra u het eenvoudige dialoogvenster hebt gemaakt, moet u de inhoud ervan definiëren. De gemeenschappelijke punten (drukknoppen, invoervakken, enz.) hebben elk hun eigen procedure. De inhoud van het dialoogvenster moet slechts eenmaal worden gedefinieerd in uw programma initialisatie fase.

Alle dialoogvensters moeten minstens één drukknop hebben, met de tekst OK, waarmee de gebruiker kan bevestigen dat zijn inbreng voltooid is:

```
PROC_pushbutton(dlg%,text$,id%,x%,y%,cx%,cy%,style%)
```

De waarde **dlg%** identificeert het dialoogvenster die teruggegeven werd door **FN_newdialog**. De string **tekst$** geeft de tekst op de knop, bijvoorbeeld “OK”, de waarden **x%** en **y%** geven de positie van de knop binnen het dialoogvenster en de waarden **cx%** en **cy%** geven de grootte van de knop (in dialoogvenster eenheden).

De waarde **id%** is een unieke identifier van de drukknop; alle items binnen een particulier dialoogvenster moeten verschillende waarden hebben. U kunt een willekeurige waarde (binnen een reden) kiezen, maar de waarden 1 en 2 zijn gereserveerd voor de knoppen **OK** en **Cancel**. Die mogen alleen worden gebruikt voor dat doel.

De waarde **style%** kan nul zijn, maar andere waarden kunnen het uiterlijk of gedrag van de drukknop wijzigen. De waarde 1 (BS_DEFPUSHBUTTON) stelt de drukknop in als de standaardknop; het wordt weergegeven met een dikkere rand en op de Enter toets te drukken heeft hetzelfde effect als het indrukken van deze knop. Er moet slechts één knop in elk dialoogvenster met deze stijl worden gemaakt; normaal zou dit de **OK** knop zijn. Het instellen van **style%** tot &20000 (WS_GROUP) zorgt ervoor dat de knop als eerste item in een nieuwe *groep* wordt; dit geldt voor het navigeren op het dialoogvenster met behulp van de cursortoetsen. Het instellen van **style%** tot &80 (BS_BITMAP) geeft aan dat de knop met een bitmapafbeelding wordt weergegeven; in dit geval zal **tekst$** leeg moeten zijn. De waarden kunnen echter gecombineerd worden.

PROC_checkbox

Een checkbox, of ook wel genoemd een selectievakje, kan telkens geselecteerd worden (bevat een klikmarker) of niet. Wanneer de gebruiker op het vakje klikt, wordt het (meestal) ingeschakeld of uitgeschakeld. Een selectievakje wordt gebruikt om een van de twee (bijvoorbeeld uit of aan) te selecteren.

```
PROC_checkbox(dlg%,text$,id%,x%,y%,cx%,cy%,style%)
```

De waarde **dlg%** identificeert het dialoogvenster en is de waarde teruggegeven van FN_newdialog. De string **text$** is de tekst die naast het selectievakje verschijnt, de waarden **x%** en **y%** is de positie van het selectievakje op het dialoogvenster en de waarden **cx%** en **cy%** is de grootte van het selectievakje en zijn geplaatste tekst, in dialoogvenster eenheden. De waarde **id%** is een unieke identifier voor het selectievakje.

De waarde **style%** kan nul zijn, maar andere waarden kunnen het uiterlijk of gedrag van het selectievakje wijzigen. Het instellen op &20 (BS_LEFTTEXT) zorgt ervoor dat de tekst links van het selectievakje wordt geplaatst, anders zal de tekst rechts van het selectievakje worden geplaatst. Het instellen op &20000 (WS_GROUP) zorgt ervoor dat het selectievakje het eerste item in een nieuwe *groep* wordt; dit is van invloed op het navigeren van het dialoogvenster bij gebruik van de cursor toetsen. De waarden kunnen gecombineerd worden.

PROC_radiobutton

Een radiobutton, ook wel genoemd een optieknop, is een klein rondje kan telkens geselecteerd worden (bevat een centrale plek) of niet. Optieknoppen worden gebruikt in groepen, van twee of meer, waarvan *alleen één* van de knoppen geselecteerd kan worden. Als de gebruiker op een van de knoppen klikt, zal die knop geselecteerd worden en alle andere knoppen in de groep zullen worden uitgeschakeld.

```
PROC_radiobutton(dlg%,text$,id%,x%,y%,cx%,cy%,style%)
```

De parameterwaarden werken op dezelfde manier als bij PROC_checkbox.

PROC_editbox

Een editbox, ook wel genoemd een invoervenster of invoerbalk, is een rechthoekig veld waar de gebruiker (alfa)numerieke invoer kan typen.

```
PROC_editbox(dlg%,text$,id%,x%,y%,cx%,cy%,style%)
```

De parameters spreken voor zich, maar **style%** heeft ander soort waarden. Wordt deze ingesteld met waarde &80 (ES_AUTOHSCROLL) dan zal de inhoud van het invoervenster horizontaal gescrolld worden, als het nodig is. Wordt het ingesteld met &2000 (ES_NUMBER) zorgt ervoor dat het invoervenster alleen maar numerieke invoer accepteert. Wordt het ingesteld met &20000 (WS_GROUP) dan zal het invoervenster het eerste item zijn in een nieuwe *groep*. Wordt het ingesteld met &1004 (ES_WANTRETURN + ES_MULTILINE), samen met een juiste verticale grootte, dan zal het invoervenster veranderen in een *multi-line* invoervenster. De waarden kunnen worden gecombineerd.

PROC_static

Een static item, ook wel genoemd een statisch item, is een rechthoekig gebied dat (meestal) een tekstinhoud heeft of een image. Het kan gebruikt worden om een ander item te labelen of een simpele informatie te geven.

```
PROC_static(dlg%,text$,id%,x%,y%,cx%,cy%,style%)
```

De parameters spreken voor zich, maar **style%** heeft weer andere waarden. Standaard is de tekst links uitgelijnd binnen de rechthoek, maar stellen we **style%** in op 1 (SS_CENTER) dan zal de tekst binnen in de rechthoek gecentreerd worden en stellen we het in op 2 (SS_RIGHT) dan zal de tekst rechts worden uitgelijnd binnen de rechthoek. Wordt het ingesteld met &E (SS_BITMAP) bepaald dat het item een bitmap zal bevatten die later ingeladen wordt; op deze manier zal **text$** leeg zijn.

PROC_listbox

Een listbox geeft een lijst weer van twee of meer items waar de gebruiker er één van kan selecteren.

```
PROC_listbox(dlg%,"",id%,x%,y%,cx%,cy%,style%)
```

De parameters spreken voor zich, maar de tekst string is ongebruikt en zal een lege string meekrijgen.

De waarde **style%** kan ingesteld worden met &20000 (WS_GROUP), dit zorgt ervoor dat de lijst als een eerste item in de groep hoort. Om het sorteren van de listbox inhoud uit te schakelen, moet u 2 (LSB_SORT) van de **style%** waarde aftrekken, anders zou het sorteren worden ingeschakeld.

De items die worden weergegeven in de keuzelijst moeten worden geschreven als een aparte oefening zodra het dialoogvenster wordt weergegeven. Zie initialisatie van de inhoud van een dialoogvenster voor details. Als de hoogte van de keuzelijst niet voor het aantal items, die moeten worden weergegeven, volstaat, wordt een verticale schuifbalk automatisch gegenereerd. Voegen we &100200 (WS_HSCROLL + LBS_MULTICOLUMN) toe aan **style%,** dan zal de keuzelijst een horizontale schuifbalk hebben en de items worden weergegeven in kolommen.

PROC_combobox

Een combobox is een lijst met een invoerbalk. De lijst geeft de opties weer dat de gebruiker kan selecteren. De geselecteerde optie komt in de invoerbalk, maar in plaats van te selecteren kan de optie ook ingevoerd worden.

```
PROC_combobox(dlg%,"",id%,x%,y%,cx%,cy%,style%)
```

De parameters spreken voor zich, maar de tekst string is ongebruikt en zal een lege string meekrijgen.

Stellen we de waarde van **style%** in met 3 (CBS_DROPDOWNLIST) dan wordt er een uitschuifbare lijst gemaakt, waarmee de lijst met items alleen wordt weergegeven als de gebruiker op de driehoek rechts van de invoerbalk klikt**.** Merk op dat met name in dit geval **cy%** de hoogte is van de uitschuifbare lijst van de combobox. Stellen we de waarde &100 in (CBS_SORT) dan zullen de items in de lijst gesorteerd worden in alfabetische volgorde. Met de waarde &20000 (WS_GROUP) wordt deze combobox het eerste item in een nieuwe groep. Deze waarden kunnen ook gecombineerd worden.

De items, die worden weergegeven in de keuzelijst, moeten worden geschreven als een aparte oefening zodra het dialoogvenster wordt weergegeven. Zie initialisatie van de inhoud van een dialoogvenster voor details. In het geval van een drop-down lijst zal voldoende hoogte voor de keuzelijst met invoervak worden bepaald wanneer het wordt weergegeven.

PROC_dlgitem

PROC_dlgitem kan worden gebruikt om items te creëren, die anders zijn waar specifieke procedures voor zijn bestemd.

```
PROC_dlgitem(dlg%,text$,id%,x%,y%,cx%,cy%,style%,class%)
```

De waarde **dlg%** identificeert het dialoogscherm en is de waarde die teruggegeven is van FN_newdialog. De waarden **x%** en **y%** bepalen de positie van het item binnen het dialoogscherm en de waarden **cx%** en **cy%** bepalen de grootte van het item (in dialoogscherm eenheden). De waarden van **text$**, **style%** en **class%** zijn afhankelijk van het itemtype. De waarde **id%** is een unieke identifier voor het item.

PROC_dlgctrl

PROC_dlgctrl is vergelijkbaar met PROC_dlgitem echter dat een klasse naam opgegeven moet worden, anders dan een numerieke waarde:

```
PROC_dlgctrl(dlg%,text$,id%,x%,y%,cx%,cy%,style%,class$)
```

Hiermee kunt u in uw dialoogvenster standaard Windows ™ besturingselementen insluiten waarvoor specifieke procedures niet worden gedaan. Voorbeelden van dergelijke controls zijn **up-down controls**, **trackbars** en **progressbars**:

Een up-down control toevoegen

U kunt een *up-down control* (met een paar pijlen voor omhoog en omlaag tellen van de numerieke waarde in een invoerbalk) gebruiken door de volgende programmaregel:

```
PROC_dlgctrl(dlg%,"",id%,0,0,0,0,&50000096,"msctls_updown32")
```

Er moet direct een numerieke waarde voor de invoerbalk volgen die naast de up-down pijlen verschijnt. De up-down control positioneert automatisch zelf aan de rechterkant van de invoerbalk. U ziet dat de positiewaarden ongebruikt zijn en opgegeven worden met 0,0.

Om een bereik van de up-down control in te stellen:

```
UDM_SETRANGE = 1125
SYS "SendDlgItemMessage", !dlg%, id%, UDM_SETRANGE, 0, (min% << 16) +
max%
```

Vergeet niet dat u dit pas kan doen *na* het aanroepen van **PROC_showdialog**.

Een trackbar toevoegen

U kunt een *trackbar* (schuifbalk) aan het dialoogvenster toevoegen met gebruik van onderstaande programmaregel:

```
PROC_dlgctrl(dlg%,"",id%,x%,y%,cx%,cy%,&50000000,"msctls_trackbar32")
```

Het voorbeeld creëert een horizontale trackbar zonder drukknoppen (tick-marks). Om de stijl van de trackbar te wijzigen, voegt u één of meerdere waarden toe aan de waarde **&50000000**:

Stijl	**Naam**	**Effect**
1	TBS_AUTOTICKS	Geeft de drukknoppen weer
2	TBS_VERT	Verticale trackbar
4	TBS_LEFT	Drukknop bovenaan of links (standaard is onderaan of rechts)
8	TBS_BOTH	Drukknoppen aan beide kanten

Om een bereik van de trackbar in te stellen:

```
TBM_SETRANGE = 1030
SYS "SendDlgItemMessage", !dlg%, id%, TBM_SETRANGE, 1, (max% << 16) +
min%
```

Vergeet niet dat u dit pas kan doen *na* het aanroepen van **PROC_showdialog**.

Om de huidige positie van de trackbar in te stellen:

```
TBM_SETPOS = 1029
SYS "SendDlgItemMessage", !dlg%, id%, TBM_SETPOS, 1, position%
```

Om de huidige positie van de trackbar uit te kunnen lezen:

```
TBM_GETPOS = 1024
SYS "SendDlgItemMessage", !dlg%, id%, TBM_GETPOS, 0, 0 TO position%
```

Een progressbar toevoegen

U kunt een *progressbar* (voortgangsbalk) aan een dialoogvenster toevoegen met gebruik van de volgende programmaregel:

```
PROC_dlgctrl(dlg%,"",id%,x%,y%,cx%,cy%,&50000000,"msctls_progress32")
```

Het voorbeeld creëert een horizontale progressbar. Om de stijl van de progressbar te wijzigen, voegt u één of meerdere waarden toe aan de waarde **&50000000**:

Stijl	Naam	Effect
1	PBS_SMOOTH	Uitgestreken (anders dan een gesegmenteerde) voortgangsbalk
4	PBS_VERTICAL	Verticale voortgangsbalk

Om een bereik van de voortgangsbalk in te stellen:

```
PBM_SETRANGE = 1025
SYS "SendDlgItemMessage", !dlg%, id%, PBM_SETRANGE, 0, (max% << 16) +
min%
```

Vergeet niet dat u dit pas kan doen *na* het aanroepen van **PROC_showdialog**:

Om de huidige positie van de voortgangsbalk in te stellen:

```
PBM_SETPOS = 1026
SYS "SendDlgItemMessage", !dlg%, id%, PBM_SETPOS, position%, 0
```

Om de voortgangsbalk te laten ‘lopen’:

```
PBM_STEPIT = 1029
SYS "SendDlgItemMessage", !dlg%, id%, PBM_STEPIT, 0, 0
```

PROC_showdialog

Nadat de positie, de grootte en de inhoud van het dialoogvenster gedefinieerd is, mag het op het scherm weergegeven worden. Zolang de creatie van het dialoogvenster en de definitie van de inhoud eenmaal is gedaan, mag het venster in zoveel meerdere keren worden weergegeven als u zelf wilt:

```
PROC_showdialog(dlg%)
```

De waarde **dlg%** identificeert het dialoogvenster en is de teruggegeven waarde van FN_newdialog. Zodra het dialoogvenster weergegeven is, kunt u berichten verzenden die van invloed zijn op de inhoud ervan en om informatie op te vragen over de huidige inhoud ervan.

De inhoud van een dialoogvenster initialiseren

In veel gevallen is de inhoud aanvankelijk van het dialoogvenster wanneer de items binnen het venster worden gemaakt. Bijvoorbeeld, de inhoud van een invoervenster kan opgegeven worden in de PROC_editbox procedure aanroep. Maar u kunt ook de inhoud op andere momenten wijzigen, in het geval van keuzelijsten en keuzelijsten met invoervakken, die meerdere items bevatten. U moet dan andere methoden gebruiken om de inhoud te kunnen initialiseren. *Merk op dat deze initialisatie **na** de aanroep van PROC_showdialog plaats moet vinden.*

Om de tekst met een dialoogvenster item te associëren (zoals bijvoorbeeld de inhoud van een invoerbalk of een label voor een keuzebalk) kunt u gebruik maken van de SetDlgItemText API aanroep:

```
SYS "SetDlgItemText", !dlg%, id%, text$
```

De waarde **!dlg%** (let op het uitroepteken) is de *handle* van het dialoogvenster, die opgenomen is in het geheugen met het adres die door FN_newdialog is teruggegeven. De waarde **id%** is de identifier voor het opgegeven item en **text$** is de nieuwe tekststring die geassocieerd wordt met het item.

Is het item een invoerbalk voor numerieke invoer, dan kan de waarde in de balk worden weergegeven met SetDlgItemInt:

```
SYS "SetDlgItemInt", !dlg%, id%, value%, signed%
```

De **value%** is de nieuwe waarde om weer te geven en **signed%** bepaald of de waarde geïnterpreteerd moet worden met teken (1) of zonder teken (0).

Als de waarde gecontroleerd wordt door een up-down control, kunt u het toegestane bereik als volgt instellen:

```
UDM_SETRANGE = 1125
SYS "SendDlgItemMessage", !dlg%, id%, UDM_SETRANGE, 0, (min% << 16) +
max%
```

waarvan **id%** is de identifier voor de up-down control, **min%** is de laagste toegestane waarde en **max %** is de hoogste toegestane waarde.

Om een bitmap afbeelding in een statische item of op een drukknop in te laden, heeft u het volgende programmasegment nodig. Voor de drukknop keuze, verander `STM_SETIMAGE(370)` in `BM_SETIMAGE(247)`:

```
LR_LOADFROMFILE = 16
STM_SETIMAGE = 370
SYS "LoadImage", 0, bmpfile$, 0, cx%, cy%, LR_LOADFROMFILE TO hbitmap%
SYS "SendDlgItemMessage", !dlg%, id%, STM_SETIMAGE, 0, hbitmap%
```

De waarde **bmpfile$** is de naam van een Windows ™ bitmapbestand die de afbeelding bevat, **cx%** en **cy%** zijn de afmetingen van de afbeelding in pixels, **!dlg%** is de *handle* van het dialoogvenster en **id%** is de identifier van het statische item of drukknop in kwestie. Zodra u klaar bent met het dialoogvenster (maar niet eerder) verwijder dan de bitmap handle als volgt:

```
SYS "DeleteObject", hbitmap%
```

Om een aantal strings in een keuzelijst in te voeren, doet u het volgende:

```
LB_ADDSTRING = 384
SYS "SendDlgItemMessage", !dlg%, id%, LB_ADDSTRING, 0, "Listbox item 0"
SYS "SendDlgItemMessage", !dlg%, id%, LB_ADDSTRING, 0, "Listbox item 1"
SYS "SendDlgItemMessage", !dlg%, id%, LB_ADDSTRING, 0, "Listbox item 2"
...
```

waarvan de waarde van **id%** is de identifier voor de keuzelijst. De keuzelijst zal (standaard) sorteren de snaren in alfabetische volgorde, zodat de volgorde waarin ze zijn verzonden niet belangrijk is. Om

te schakelen sorteren aftrekken 2 (LBS_SORT) van de stijl % waarde u anders zou hebben gebruikt (Zie PROC_listbox).

Om de inhoud van een keuzelijst te kunnen legen, doet u het volgende:

```
LB_RESETCONTENT = 388
SYS "SendDlgItemMessage", !dlg%, id%, LB_RESETCONTENT, 0, 0
```

Om een stringlijst in een combobox te plaatsen, doet u het volgende:

```
CB_ADDSTRING = 323
SYS "SendDlgItemMessage", !dlg%, id%, CB_ADDSTRING, 0, "Combobox item 0"
SYS "SendDlgItemMessage", !dlg%, id%, CB_ADDSTRING, 0, "Combobox item 1"
SYS "SendDlgItemMessage", !dlg%, id%, CB_ADDSTRING, 0, "Combobox item 2"
...
```

waarvan de waarde van **id%** is de identifier voor de combobox. Deze keer zijn de items *niet* gesorteerd, dus moeten ze gezonden worden in de volgorde zoals ze moeten verschijnen. De selectie vanuit de lijst moet worden gedaan als volgt:

```
CB_SETCURSEL = 334
SYS "SendDlgItemMessage", !dlg%, id%, CB_SETCURSEL, index%, 0
```

De waarde van **index%** bepaald welke van de items op dat moment geselecteerd is (begint vanaf 0).

Om de inhoud van een combobox te kunnen legen, doet u het volgende:

```
CB_RESETCONTENT = 331
SYS "SendDlgItemMessage", !dlg%, id%, CB_RESETCONTENT, 0, 0
```

Om de status van een set optieknoppen te bepalen, kunt u gebruiken maken van de CheckRadioButton API aanroep:

```
SYS "CheckRadioButton", !dlg%, first%, last%, id%
```

Hier is **first%** de identifier van de eerste optieknop in de groep, **last%** is de identifier van de laatste optieknop in de groep en **id%** is de identifier van de knop die u aan wilt hebben (checked).

Om de status van een keuzevak (checkbox) te bepalen, kunt u gebruik maken van de CheckDlgButton API aanroep:

```
SYS "CheckDlgButton", !dlg%, id%, state%
```

Hier is **id%** de identifier van de knop die u wilt bepalen en **state%** is de status die u wilt hebben: 0 betekent niet aangevinkt en 1 betekent wel aangevinkt.

Uitzetten of aanzetten dialoogscherm items

U kunt een of meer items in een dialoogscherm uitzetten, bijvoorbeeld een knop uitzetten als deze even niet om een bepaalde omstandigheid voldoet. Een item dat uitstaat wordt grijs weergegeven.

Om een item uit te zetten kunt u gebruik maken van de EnableWindow API aanroep:

```
SYS "GetDlgItem", !dlg%, id% TO h%
SYS "EnableWindow", h%, 0
```

Hier is **id%** de identifier van de knop die u wilt bepalen. Om deze weer aan te zetten, wijzig de nul in een één:

```
SYS "GetDlgItem", !dlg%, id% TO h%
SYS "EnableWindow", h%, 1
```

Onthoud dat, net als met de initialisatie, deze routines uitgevoerd moeten worden **na** de PROC_show-dialog aanroep.

De inhoud van een dialoogformulier lezen

Vanwege het hele werk van een dialoogscherm uit het ophalen van gebruikersinvoer bestaat, is het raadzaam de huidige inhoud te kunnen bepalen. Vooral wanneer de **OK** knop is ingedrukt.

Om de tekst te lezen die gekoppeld is met een dialoogscherm item (zoals bijvoorbeeld de inhoud van een invoervenster of een huidige selectie van een keuzelijst) kunt u gebruik maken van de GetDlgItemText API aanroep:

```
DEF FNgetdlgtext(dlg%, id%)
LOCAL text%
DIM text% LOCAL 255
SYS "GetDlgItemText", !dlg%, id%, text%, 255
= $$text%
```

De parameter **dlg%** is de teruggegeven waarde vanuit FN_newdialog en de parameter **id%** is de identifier van het doorgegeven item. De maximale lengte van de string is 255 karakters.

Voor een keuze als de *multi-line* invoerscherm kan de maximale tot een hoge waarde worden ingesteld (tot een maximum van 65535 bytes). Om de teruggegeven gegevens te bewaren in een bestand, kunt u het volgende doen:

```
DEF PROCsavetofile(dlg%,id%,filename$)
LOCAL text%, Len%
DIM text% LOCAL 65535
SYS "GetDlgItemText", !dlg%, id%, text%, 65535 TO Len%
OSCLI "SAVE """+filename$+""" "+STR$~text%+"+"+STR$~Len%
ENDPROC
```

De teruggegeven gegevens van een multi-line invoerscherm bestaat uit regels van tekst gescheiden door CRLF (CHR$13+CHR$10) op codes. Als u de gegevens wilt verwerken, kunt u het verdelen in meerdere regels als volgt:

```
P% = text%
REPEAT
      A$ = $P%         : REM. get line of text from memory
      PRINT A$         : REM. print the text (for example)
      P% += LEN(A$)+2 : REM. advance pointer to next line
UNTIL P% >= (text%+Len%)
```

Als het item een invoerscherm is met numerieke ingang, dan kan de huidige waarde gelezen worden met GetDlgItemInt:

```
SYS "GetDlgItemInt", !dlg%, id%, 0, signed% TO value%
```

De **!dlg%** waarde is de *handle* van het dialoogscherm, **id%** is de identifier voor het opgevraagde item en **signed%** bepaald of een negatieve waarde geaccepteerd mag worden (1) of niet (0).

Om te bepalen welk item (als er een is) geselecteerd is in een lijst (listbox), dan kunt u het volgende doen:

```
LB_GETCURSEL = 392
SYS "SendDlgItemMessage", !dlg%, id%, LB_GETCURSEL, 0, 0 TO sel%
```

waarvan de **id%** waarde de identifier is voor de listbox. De teruggegeven waarde **sel%** geeft de index (beginnend bij 0) van het huidig geselecteerde item. Als er geen item geselecteerd is, dan zal -1 worden teruggegeven. Merk op dat als de inhoud van de listbox gesorteerd is, wat zijn normale gedrag is, de index alleen maar weinig nut heeft. Nochtans kunt u het gebruiken om te ontdekken wat de geselecteerde tekst is, en dat kunt u als volgt doen:

```
DEF FNgetlistboxtext(dlg%, id%, sel%)
LOCAL text%
DIM text% LOCAL 255
LB_GETTEXT = 393
SYS "SendDlgItemMessage", !dlg%, id%, LB_GETTEXT, sel%, text%
= $$text%
```

Om te bepalen welk item geselecteerd is in een combobox, doet u het volgende:

```
CB_GETCURSEL = 327
SYS "SendDlgItemMessage", !dlg%, id%, CB_GETCURSEL, 0, 0 TO sel%
```

Om de huidige status van een selectievakje of keuzerondje te ontdekken, kunt u de IsDlgButtonChecked API aanroep gebruiken:

```
SYS "IsDlgButtonChecked", !dlg%, id% TO state%
```

waarvan de **id%** waarde is de identifier voor het selectievakje of keuzerondje. De waarde **state%** is 0 als de knop niet ingeschakeld is of 1 als het wel ingeschakeld is.

Bepalen wanneer een knop is aangeklikt

U moet kunnen bepalen wanneer de gebruiker op een knop heeft geklikt, bijvoorbeeld wanneer de **OK** knop is aangeklikt kan de inhoud van het dialoogformulier worden gelezen en het formulier van het scherm worden verwijderd. Wanneer u een item in een dialoogvenster aanklikt, wordt een bericht verzonden (vergelijkbaar met die van een menu of een werkbalk) dat opgespoord kan worden met ON SYS.

Het volgende codesegment kan worden gebruikt om te wachten tot op de knop **OK** of op de knop **Annuleren** wordt geklikt, en daarna vervolgens passende maatregelen te kunnen nemen:

```
Click% = 0
ON SYS Click% = @wparam% : RETURN
REPEAT WAIT 1
      click% = 0
      SWAP click%,Click%
UNTIL click% = 1 OR click% = 2 OR !dlg% = 0
ON SYS OFF
IF click% = 1 THEN
      PRINT "OK pressed"
```

```
        REM. process contents of dialogue box here
ELSE
        PRINT "Cancel pressed"
ENDIF
PROC_closedialog(dlg%)
```

Zodra op de sluitknop van een dialoogformulier (of zwevende werkbalk) geklikt wordt, produceert dat altijd dezelfde ID code (2); u kunt niet direct aan ON SYS vertellen welk dialoogformulier of werkbalk gesloten was. Als uw programma meer dan een tegelijkertijd open heeft staan, dan is dit mogelijk een probleem. U kunt bepalen welke nog open zijn, (dus bij een eliminatieproces, welke gesloten werd) door onderzoek van het venster handle (**!dlg%** in bovenstaand voorbeeld) . Als de handle niet nul is dan is het venster nog geopend, en als het nul is dan is deze gesloten.

PROC_closedialog

Wanneer de gebruiker op **OK** heeft geklikt en de inhoud van het dialoogformulier verwerkt heeft, kan het formulier (algemeen) van het scherm worden verwijderd:

```
PROC_closedialog(dlg%)
```

Het dialoogformulier *sjabloon* blijft in het geheugen, zodat u het altijd weer kunt aanroepen met PROC_showdialog.

Het dialoogformulier zal ook worden verwijderd wanneer uw programma terug zal schakelen in directe mode (bijvoorbeeld als een foutmelding verschijnt of het END statement uitgevoerd wordt) of wanneer het scherm van uw programma gesloten wordt door de gebruiker. Dit kan worden bereikt door het uitvoeren van de volgende statements onmiddellijk na de aanroep van PROC_showdialog:

```
ON CLOSE PROC_closedialog(dlg%):QUIT
ON ERROR PROC_closedialog(dlg%):PRINT'REPORT$:END
```

Omdat het dialoogformulier ruimte gebruikt op de *heap*, is het essentieel dat u het verwijderd voordat u een CLEAR, CHAIN of RUN statement uitvoert. Doet u dit niet, dan is het zeer waarschijnlijk dat *BBC BASIC for Windows* kan crashen.

API’s voor Liberty BASIC.

API’s aanroepen

Elke Windows programmeertaal roept API functies aan. Deze functies zijn een integraal onderdeel van het Windows besturingssysteem. Ze zijn opgenomen binnen in bestanden die Dynamic Link Library’s worden genoemd. Deze DLL bestanden bevatten functies die beschikbaar zijn voor programmeurs. API staat voor Application Programming Interface. De API beschrijft hoe de functies toegankelijk zijn in een DLL.

Liberty BASIC roept veel Windows DLL’s aan achter de schermen wanneer native functies en commando’s worden gebruikt in een programma. Een voorbeeld: om een formulier te openen, zal de functie CreateWindowEx worden aangeroepen op de achtergrond in Liberty BASIC.

Windows gebruikt een manier om devices en objecten te kunnen handelen. Wanneer Liberty BASIC een formulier creëert, geeft het een handle aan Windows. Wanneer Liberty BASIC een control (listbox, button, textbox, enz.) creëert, wordt er ook een handle gegeven. Wanneer Liberty BASIC een bestand opent, geeft het een handle. Een Windows handle is een nummer. Het is een uniek nummer dat het

formulier, control of ander apparaat identificeert. Geen ander apparaat heeft hetzelfde nummer (handle) als een ander in werkende sessie van Windows.

In Liberty BASIC koppelt een programmeur een bestand met zoiets als: #myfile en voor toegang met een formulier zoiets als: #mywindow. Dit zijn Liberty BASIC handles, geen Windows handles. Liberty BASIC heeft een functie die de Windows handle van een gegeven Liberty BASIC formulier of control teruggeeft. Het is de HWND functie, met een Open commando zoals dit:

```
button #w.exit, "Exit", [Close.w], UL, 200, 300, 60, 25
open "My Window" for window as #w
```

het is mogelijk om de Windows handle voor het formulier te krijgen met:

```
hw = hwnd(#w)
```

en voor de 'Exit' knop binnen het formulier:

```
hwExit = hwnd(#w.exit)
```

Wanneer bij een API aanroep de handles vereist zijn, kunt u gebruik maken van de teruggegeven Windows handle van de HWND() functie. Het handelen van de argumenten in API functies moet altijd worden gedaan met het type ULONG, dat een 32-bit integer waarde is zonder teken.

CALLDLL
Gebruik CallDll om een API functie aanroep te maken in Liberty BASIC. Voordat u CallDll gebruikt, moet u het passende DLL openen voor gebruik en geeft het een Liberty BASIC handle, zoals dit:

```
Open "DLLname.dll" for dll as #dllname
```

Bepaalde Windows besturingssysteem DLL's vereisen niet een OPEN statement voor gebruik met CALLDLL. De volgende DLL's mogen gebruikt worden zonder ze eerst te openen. Probeer niet een handle te sluiten, tenzij ze geopend zijn in de programmacode. Deze DLL's werken wel met OPEN, maar het is niet langer meer nodig:

#user32	#kernel32	#gdi32	#winmm
#shell32	#comdlg32	#comctl32	

Zodra de DLL geopend is, geeft het een handle voor gebruik bij de CallDll functie. Elke toegestane naam mag gebruikt worden voor die handle, maar de code is meer gemakkelijk te begrijpen en te hergebruiken als bepaalde conventies er op volgen. Gebruik een versie van de DLL's naam voor de leesbaarheid. Bijvoorbeeld, als u URLMON.DLL opent dan kunt u het openen als `#urlmon` of als `#url`.

Wanneer de DLL niet langer meer nodig is, moet die gesloten worden met het CLOSE commando.

```
Close #urlmon
```

Probeer geen DLL handle te sluiten die niet geopend is in een programma. De DLL zal elke keer geopend worden als een functie is aangeroepen en daarna onmiddellijk worden gesloten. Als een DLL in het hele programma in gebruik is, dan is het gemakkelijker en efficiënter om het eenmalig bij de start van het programma te openen en weer te sluiten wanneer het programma beëindigd wordt.

Eén regel
Het CALLDLL statement moet in één regel geschreven worden. Liberty BASIC ondersteunt het gebruik van het onderstrepingsteken (underscore) om een regel te breken, zodat het leesbaar kan wor-

den in de editor. Elk CALLDLL statement is uniek, omdat elke API aanroep verschillende nummers en types van parameters vereist. Hier is een algemeen voorbeeld:

```
'een één regel:
CallDll #handle, "FunctieNaam", Parm as TYPE, Parm2 as TYPE, Result as
        ReturnTYPE

'weergegeven in meerdere regels door gebruik van underscores _:
CallDll #handle, "FunctieNaam",_
     Parameter1 as TYPE,_
     Parameter2 as TYPE,_
     Result as ReturnTYPE
```

Het voorbeeld roept een functienaam in een DLL aan met twee parameters. Het geeft een waarde terug in de variabele Result.

Argumenten
We kunnen de argumenten, of parameters, zien als een lijst van instructies. De functie die aangeroepen wordt bevat acties gebaseerd op die instructies vanuit het programma. De parameter (instructie) gegeven in de MessageBeep functie vertelt welke soorten geluiden gemaakt moeten worden.

De argumenten van de functies moeten in de juiste volgorde staan en het elk het juiste TYPE hebben.

Argumenten kunnen geen array elementen zijn. CALLDLL kan geen expressies gebruiken in de argumenten. De volgende twee voorbeelden zijn beide onjuist en zullen niet werken.

```
CallDll #handle, "FunctieNaam",_
     (Parameter1 + Parameter2) as TYPE,_    'kan geen expressies gebruiken
     Result as ReturnTYPE

CallDll #handle, "FunctieNaam",_
     array(3) as TYPE,_       'kan geen array elementen gebruiken
     Result as ReturnTYPE
```

Types in CALLDLL
Liberty BASIC heeft geen numeriek getypeerde variabelen, maar API functies vereisen dat deze parameters getypeerd zijn. Het “AS TYPE” deel van de CALLDLL functie stelt Liberty BASIC in staat het argument in het TYPE te converteren bij de functie. De numerieke types, die 2 bytes (16 bits) lang zijn, doen onder andere het volgende:

```
SHORT
USHORT
WORD
```

De volgende numerieke types zijn 4 bytes (32 bits) long:

```
LONG
ULONG
DOUBLE
```

Getallen met fractionele onderdelen, zoals een aantal decimalen achter de komma, kunnen niet worden doorgegeven in API aanroepen.

De volgende types zijn 4 bytes lang, maar ze zijn geen numerieke types. Het zijn pointers naar een geheugendeel waar informatie door een programmeur werd opgeslagen. Later worden deze ook uitgelegd:

```
STRUCT
PTR
```

Sommige functies geven geen waarde terug, maar Liberty BASIC verwacht altijd dat er een waarde teruggegeven wordt. Het type voor de API functies die geen waarde terug geven is:

```
VOID
```

Het boolean type wordt gebruikt wanneer een argument waar (true) of onwaar (false) is. Is de waarde 0 dan is het FALSE, anders is het TRUE.

```
BOOLEAN
```

Hier is een actuele API aanroep. Dit is een compleet programma. Het roept de functie vanuit user32.dll om een MessageBeep te maken. De instructie “32 as long” vertelt Liberty BASIC om het getal 32 te nemen en door te geven aan de functie als een 32 bit getal. 32 is de instructie om een geluid te maken dat geassocieerd is met een mededeling aan het gebruikerssysteem.

```
CallDll #user32, "MessageBeep",_
     32 as long, result as boolean
```

Letterlijke waarden en variabelen in CALLDLL
Argumenten kunnen letterlijke waarden of tekenreeksen zijn, of het kunnen variabelen zijn. Het bovenste voorbeeld kon net zo goed geschreven worden als dit; met een variabele in plaats van een letterlijke waarde voor het argument:

```
Sound = 32
CallDll #user32, "MessageBeep",_
     Sound as long, result as boolean
```

Teruggave
Sommige functies geven een waarde terug. Eén van de functies is GetTickCount, die het aantal milliseconden van de tijd teruggeeft die verstreken is vanaf de huidige startsessie van Windows. Sommige functies geven geen waarde terug. Liberty BASIC verwacht altijd een teruggegeven waarde. Geeft een functie geen waarde terug, dan zal het ReturnTYPE automatisch een VOID teruggeven.

Windows constanten
Liberty BASIC kan veel Windows constanten erkennen. Als de Liberty BASIC compiler een voor gedefinieerde Windows constante tegenkomt, wordt de numerieke waarde van die constante genomen. Constanten maken het gemakkelijk voor de programmeurs om de instructies leesbaar te houden, en ook zodat programmeurs gemakkelijk met verschillende talen van de een naar het ander kan communiceren. De “32” gebruikt in de MessageBeep functie is de waarde voor de Windows constante, MB_ICONEXCLAMATION. Deze constante gebruiken is gemakkelijker te onthouden voor de Message Beep Exclamation dan de waarde “32” te onthouden. De mogelijke Windows constanten die gebruikt kunnen worden in de MessageBeep functie, staan hieronder met hun eigen numerieke waarden:

```
MB_ICONASTERISK     = 64      MB_ICONQUESTION     = 48
MB_ICONEXCLAMATION  = 32      MB_ICONHAND         = 16
MB_OK               =  0
```

Om gebruik te maken van de Windows constanten in Liberty BASIC, moet u een underscore "_" voorvoegsel voor een constante om Liberty BASIC te vertellen dat dit een voor gedefinieerde Windows constante is. De constanten verschijnen op deze manier in Liberty BASIC syntax:

```
_MB_ICONASTERISK    _MB_ICONQUESTION    _MB_ICONEXCLAMATION
_MB_ICONHAND        _MB_OK
```

Windows constanten mogen gebruikt worden in plaats van hun soortgelijke waarden. Een simpel programma, dat een constante gebruikt in plaats van een "32" waarde, doet dat zoals hieronder:

```
CallDll #user32, "MessageBeep",_
     _MB_ICONEXCLAMATION as long, result as boolean
```

Een Windows constante mag ook toegekend worden aan een variabele en op die manier toegepast worden:

```
Sound = _MB_ICONASTERISK
CallDll #user32, "MessageBeep",_
     Sound as long,_
     Result as boolean
```

Meer dan één constante gebruiken
Soms is het nodig om meerdere Windows constanten te combineren voor gebruik in API aanroepen. Dit wordt gedaan met de bitgewijze "OR" operator. Combineer de constanten in de lijst met het woord "OR" ertussen. Hier is een voorbeeld met gebruik van Window Stijl constanten die gebruikt worden in verschillende API aanroepen om vensters en controls te creëren:

Meerdere constanten met "OR":
```
style = _WS_VISIBLE or _WS_CHILD
```

Negatieve nummers in CALLDLL
Liberty BASIC ondersteunt negatieve nummers als argumenten in API aanroepen.

Struct
Structs worden gebruikt in API aanroepen om een lijst met informatie te verzenden en/of een lijst met informatie van de functie te ontvangen. Onthoud, wanneer een argument doorgegeven wordt aan een API aanroep AS STRUCT, dat een 32 bit geheugenadres doorgegeven wordt, maar niet de informatie zelf. Een functie kan alleen maar één waarde teruggeven met result, dus als meerdere waarden teruggegeven moeten worden, is een struct nodig.

Een struct wordt gemaakt door het een naam te geven en daaropvolgend de members op te geven, elk met AS TYPE, net zoals in het CALLDLL statement. Een struct mag één of meerdere members hebben. Hier is de syntax:

```
Struct myStruct, eerste as type, tweede as type, derde as type
```

Een struct wordt weergegeven in één regel, zoals een API aanroep, maar het mag afgebroken worden in meerdere regels met het underscore symbool:

```
Struct myStruct,_
eerste as type,_
tweede as type,_
derde as type
```

Een struct moet gedeclareerd worden voordat het in een API aanroep gebruikt kan worden. In sommige gevallen is het nodig om waarden aan de members van de struct toe te kennen voordat de API aangeroepen kan worden, zodat die de nodige instructies bevat voor de API functie. De struct voor de x en y coördinaten van een punt is soms nodig. Hier is een voorbeeld:

```
STRUCT Point, x as long, y as long
```

Het kan ook op deze manier geschreven worden:

```
STRUCT Point,_
x as long,_
y as long
```

Structs moeten net zo een naam hebben als normale variabelen die hebben, maar voeg geen punt in een naam en gebruik geen woorden die net zo worden genoemd als de Liberty BASIC commando's en functies. Deze worden gereserveerde woorden genoemd (sleutelwoorden).

Structs zijn geen types
Sommige andere programmeertalen hebben TYPES, die lijken eruit te zien als echte STRUCTS. STRUCTS zijn niet hetzelfde als TYPES in andere programmeertalen. Een STRUCT is een enkele instantie van een structuur. Het kan alleen toegankelijk worden gemaakt door gebruik van de naam van de STRUCT.

```
'Juiste Liberty BASIC code:
STRUCT Point, x as long, y as long
Point.x.struct = 11
Point.y.struct = 52

'Onjuiste Liberty BASIC code die wel in andere programmeertalen werkt:
TYPE Point, x as long, y as long
DEF myPoint as Point
myPoint.x = 11
myPoint.y = 52
```

Waarden toekennen aan structs
Zie hier hoe een Point struct gevuld wordt, voordat het gebruikt wordt in een API aanroep:

```
Point.x.struct = 33
Point.y.struct = 124
```

De volgorde is: de structnaam gevolgd door een punt en de member (hier is het x of y) dan nog een punt en als laatst het woord "struct". Vul een struct bij toekenning van een waarde aan een member met gebruik van een = symbool. De WindowFromPoint functie gebruikt een struct om een punt te zetten op het scherm. De functie geeft de handle terug van het venster op het scherm die het geplaatste punt heeft. Na het declareren van de Point struct en het toekennen van de waarden voor de members, kan het in een functie worden gebruikt, zoals dit:

```
CallDll #user32, "WindowFromPoint", Point as struct, winHandle as ulong
```

De variabele winHandle heeft de handle van het venster die het punt bevat in de struct.

Waarden terugkrijgen vanuit structs
Structs mogen gevuld worden met waarden bij een API functie en deze waarden kunnen teruggegeven worden bij gebruik van een API functie. De aanroep naar GetCursorPos gebruikt een Point struct zoals een gebruikt wordt in het WindowFromPoint voorbeeld. De struct moet gedeclareerd zijn voordat

het aangeroepen wordt. De waarden voor de struct members hoeven niet toegekend te worden voordat het aangeroepen wordt. De functie kent de waarden toe aan de struct members.

```
CallDll #user32, "GetCursorPos", Point as struct, result as long
```

Om de x, y positie van de muiscursor te bepalen na de aanroep, kunt u de waarden uit de struct halen, zoals dit:

```
Xcursor = Point.x.struct
Ycursor = Point.y.struct
```

De coördinaten van de muiscursor zijn nu opgeslagen in de variabelen Xcursor en Ycursor.

Een member van een struct mag direct worden gebruikt, zonder het toe te kennen aan een variabele. Hier zijn enkele voorbeelden:

```
Print Point.x.struct
Xytotal = Point.x.struct + Point.y.struct
Print SIN(Point.x.struct)
Number$ = STR$(Point.y.struct)
```

Sommige functies gebruiken informatie gekoppeld binnen een struct. De struct wordt gevuld met verschillende waarden die de programmeur terug mag krijgen. De struct kan worden gebruikt bij een API functie voor invoer van de waarden, het uitlezen van de waarden of beide.

Er zijn zeldzame instanties wanneer het nodig is een array van bytes (karakters) door te geven in een API aanroep als deel van een struct. Gebruik het speciale type CHAR[n] voor dit doel. Plaats het aantal bytes of karakters tussen de rechte haken.

```
CHAR[n]
```

Het is mogelijk en soms nodig om een struct door te geven als een member van een andere struct. Dit voorbeeld gebruikt beide van deze technieken:

```
STRUCT BITMAPINFOHEADER,_
          biSize As Long,_
          biWidth As Long,_
          biHeight As Long,_
          biPlanes As Short,_
          biBitCount As Short,_
          biCompression As Long,_
          biSizeImage As Long,_
          biXPelsPerMeter As Long,_
          biYPelsPerMeter As Long,_
          biClrUsed As Long,_
          biClrImportant As Long

Struct BITMAPINFO,_
          BITMAPINFOHEADER as struct,_
          bmiColors As char[256]
```

Hier is een voorbeeld van Carl Gundel die laat zien hoe u de members van een struct vult en toegang geeft dat ook een member is van een andere struct:

```
'definieer twee structs:
     struct range, min as long, max as long
     struct fill, string as ptr, range as struct

'vul de "range" struct:
     range.min.struct = 50
     range.max.struct = 75

'vul de "string" member van de "fill" struct:
     fill.string.struct = "some string"

'sla de range struct in de fill struct op:
     fill.range.struct = range.struct

'roep een mogelijke API aan
'haal de range struct eruit:
     range.struct = fill.range.struct

'het is nu mogelijk toegang te krijgen tot de waarden:
     print range.min.struct
     print range.max.struct

'het is ook mogelijk om een andere struct te gebruiken om de waarden
'te krijgen:
     struct another, lower as long, higher as long
     another.struct = fill.range.struct
     print another.lower.struct
     print another.higher.struct
```

Werken met formules in expressies.

Formules gebruiken we om berekeningen uit te voeren in programma's die code uit moeten voeren waar geen functies voor zijn gemaakt. Zou u wel eens een boekhouding willen maken? Nee, niet met gebruik van een al voorgemaakt boekhoudprogramma, maar met uw eigen programma met alleen uw eigen functionele zaken.

Elke boekhouding is weer anders. Hoe zo'n project gemaakt wordt, valt helaas buiten dit hoofdstuk van het bulletin. Maar misschien dat ik eens een keer een voorbeeld laat zien.

Vanzelfsprekend werkt een voorgemaakt boekhoudprogramma in zijn algemeen. Het is openlijk en voor elke zaak bestemd. Om het toch beperkt te houden voor elk bedrijf, bestaan er sjablonen. Kijk maar eens naar Office. Alle programma's van Office hebben ook sjablonen. Daar zou meestal elk bedrijf met hun eigen gereedschap mee uit de voeten kunnen.

Zelf formules schrijven en opbouwen
Programma's worden opgebouwd uit codeblokken. Zo'n codeblok kan een unit zijn die uit vele functies bestaat. Door de formules in functies te programmeren en ze in aparte units op te bouwen, maken we op die manier een eigen motor voor het hoofdprogramma.

Een formule is geen functie
Een formule kan een functie zijn, maar andersom niet altijd. Een functie kan code bevatten die helemaal niet als een formule kan werken. Daarom is een formule geen functie.

Het oude statement: DEF FNn(x) = m
Het statement **DEF FN** heeft een lange baard. In de oude microcomputers met BASIC versies, zoals de Commodore VIC 20, de Commodore 64 en de Commodore 128, werd dit statement veel gebruikt om formules als functies in BASIC uit te kunnen voeren. Na de tijd met GW-BASIC werd het statement snel vergeten, want QBASIC kwam met betere methoden om sub- en functieblokken te kunnen schrijven.

Een formule is geen expressie
Net als een formule geen functie is, is een formule ook geen expressie.
De reden dat het bij beide het geval is, komt doordat formules wiskundig geschreven worden. Ook een som is een formule, maar zodra er een functie in zit dan is het een expressie. Er is echter nog een reden waarom we formules simpelweg gewoon niet in de code direct kunnen uitvoeren:

- (On)gelijknamige vergelijkingen:
  5x - 12 = 3x + 4
- Kwadraten
  $x^2 – 18 = -2$
- Technische formules, o.a.
  2x(x – 8) = 0
  *x = β * 2π*
  *v = 4π * Cos(x)*

Om zulke formules toch in expressies uit te kunnen voeren, zouden ze ontleed moeten worden. We kunnen niet zomaar 2x(x – 8) = 0 gaan schrijven. De compiler zou niet kunnen begrijpen wat hier bedoeld wordt. Om goede programma's te kunnen schrijven die wiskundige formules uit moeten voeren, moeten ze omgezet worden. In sommige gevallen kan dat erg lastig zijn. Hieronder heb ik wat oplossingen:

- De vergelijkingen kunnen opgelost worden door *x* alleenstaand voor de = te krijgen, zie later.
- Staat er een getal met erachter een letter, zoals: 5x, maak er dan van: 5 * x.
- Staat er een letter met een klein cijfertje, zoals: x^2, maak er dan van: x ^ x, een berekening die machtsverheffen wordt genoemd.
- Staat er uit het vorige punt nog een getal ervoor, dan heeft machtsverheffen een hogere prioriteit dan vermenigvuldigen. Na x ^ x te hebben berekend, moet dit vermenigvuldigd worden met het getal, $2x^2$ wordt 2 * x ^ x.

De technische formules zijn het moeilijkst en vaak is het niet mogelijk het om te zetten. Bij zulke formules worden wel functies geschreven die een omweg maken, zodat het resultaat toch juist uitkomt.

Hebben we de juiste functie voor een formule kunnen maken? Prima, maar alleen de functie van de formule is niet altijd voldoende zoals onderstaand kwadraat laat zien:

$$X^2 + 4 = 0$$

Hoewel deze formule onschuldig lijkt, kan deze nooit uitgevoerd worden. Ook niet als we er een functie van maken. Converteren we deze naar een functie, dan zal de volgende som uitkomen:

$$X * X = - 4$$

Of we nu X als waarde 2 proberen of -2, uit beide zal het antwoord nooit -4 kunnen zijn.

Een IF … THEN conditie zal hieruit moeten bepalen of de waarde voor de = positief of negatief is. Negatieve waarden zullen altijd uit bovenstaande formule prima uitgevoerd worden, omdat de waarde positief wordt zodra we die verhuizen achter de =.

Wiskundigen zullen onderstaande regel maar een rare formule vinden:

A = A = 10

Dit is eigenlijk geen formule, maar een conditie waarvan het resultaat wordt toegekend aan dezelfde variabele. Vergist u zich hiermee niet: A krijgt niet de waarde 10, maar er wordt gecontroleerd of A die waarde 10 heeft. Zo ja, dan zal A een nieuwe waarde krijgen, namelijk *waar* (niet 0) of *onwaar* (wel 0). In sommige BASIC dialecten is dit toegestaan, maar als het niet zo is dan moeten we dezelfde regel met twee verschillende variabelen schrijven:

B = A = 10

Is variabele B van het type Boolean, dan zal het zonder problemen werken. Is A gelijk aan 10, dan wordt variabele B *true*, anders *false*.

Een voorbeeld van een formule die we gemakkelijk kunnen converteren, kan een wiskundige vergelijking zijn. Zien we een formule als deze:

x – 4 = 30 – x

dan kunnen we die in een functie zo converteren:

x = 34 – x

Kijk eens naar een volledige BASIC functie:

```
Function Vergelijking(ByVal N1 As Integer, ByVal N2 As Integer) As Integer
     Dim X As Integer

     'Vergelijking: x + n1 = n2 - x
     X = -N1 + N2 - X
     Vergelijking = X
End Function
```

In stappen uitvoeren
Een bekende expressie die we in code uit kunnen voeren, maar in de wiskundetaal niet bestaat is optellen of aftrekken in het aantal opgegeven stappen, bijvoorbeeld:

```
X = X + 5
```

In de wiskunde wordt dit gezien als een gelijknamige vergelijking. De X achter de = moet voor de = staan, zodat we een formule krijgen als:

X – X = 5

Omdat we nooit twee verschillende waarden aan dezelfde X kunnen geven, moeten we dit zo berekenen dat we met twee dezelfde waarden als resultaat op 5 uit moeten komen, en u zult het meteen al zien: op dat resultaat zal men nooit komen, want het antwoord zal altijd 0 zijn.

Daarom is het geen code die we moeten proberen te converteren naar een functie. Dit is een echte computercode regel. Elke programmeertaal kent deze manier van het elke keer optellen van een waarde met een variabele en het resultaat toekennen aan dezelfde variabele.

Verwarrend? We apen Delphi na!
In VB .NET kunnen we die regel korter schrijven:

```
X += 5
```

Voor mensen die gewone basis BASIC gebruiken of VBA, zullen deze regel niet kennen. Het werkt wel op dezelfde manier. Kijk eens naar onderstaande twee Delphi statements:

```
Inc(X, 5);
Dec(Y, 3);
```

Het Inc statement (Increment = ophogen) werkt op dezelfde manier als X = X + 5, en het Dec statement (Decrement = verlagen) werkt als: Y = Y – 3.

Veel gemakkelijker te begrijpen? Dan heb ik wat voor u. We kunnen deze twee Delphi statements in BASIC namaken, zodat u geen last meer hebt van het constant herhalen van de variabele in de regel. Hieronder staan ze, gebruik ze in elk programma dat u schrijft:

```
Sub Inc(ByRef X As Integer, ByVal S As Integer)
     X = X + S
End Sub

Sub Dec(ByRef X As Integer, ByVal S As Integer)
     X = X - S
End Sub

'Hier een voorbeeld om een relatieve waarde 5 op te hogen:
Inc X, 5

'Let op, gebruik geen expressie in de ByRef parameter:
Inc N + 10, 1    'Werkt niet

'Een voorbeeld om te verlagen, met een stap-expressie:
Dec I, N + 2
```

Een ander statement dat we kunnen maken is:

```
IfStep X, 5, 10
```

Dit statement moet het volgende doen: 10 keer met 5 stappen in X ophogen. Onderstaande code laat zien hoe we het maken:

```
Sub IfStep(ByRef X As Integer, ByVal S As Integer, ByVal A As Integer)
     Dim I As Integer

     For I = 1 To A
          'Er zijn twee mogelijkheden die we kunnen gebruiken:

          'Mogelijkheid 1: Een korte methode, één regel.
          X = X + S
```

```
          'Mogelijkheid 2: De Delphi statements gebruiken, meerdere regels
          'zijn wel nodig.
          If S < 0 Then
               Dec X, S
          Else
               Inc X, S
          End If

     Next I
End Sub
```

Waarom controleren we variabele S niet in mogelijkheid 1? De reden is dat een negatieve waarde altijd voorrang heeft. Tellen we bijvoorbeeld:

```
     X = X + A
```

Dan zal met `A = -5` hetzelfde gebeuren:

```
     X = X - 5
```

Dus een regel: `Inc N, -1` werkt hetzelfde als: `Dec N, 1`. Maar toch is het geen slecht idee om deze twee berekeningen uit elkaar te houden. Wilt u verlagen? Tel dan niet op met een negatieve waarde. Dat maakt de code verwarrend en ook voor u. Maak anders gebruik van een If … Then statement in de Inc subroutine die controleert of de parameter S positief is. Zo ja, dan mag deze opgeteld worden.

Wat betreft het IfStep statement, dit is geen Delphi statement. Ik heb het statement zelf bedacht.

Een andere functie die handig is, is de IsBetween functie. Hieronder staat de code:

```
Function IsBetween(ByVal X As Integer, ByVal S As Integer, _
                   ByVal E As Integer) As Boolean
     IsBetween = (E > X) And (E < X + S)
End Function
```

Op die manier kan worden bepaald of de eindwaarde E tussen de beginwaarde X en de beginwaarde plus de stap ligt, bijvoorbeeld:

```
B = IsBetween(50, 15, 100)    'Deze functie zal een False geven aan B.
B = IsBetween(N, 10, 40)      'Zal True zijn als 40 tussen de N en N + 10
                              'ligt. Het ligt eraan wat de beginwaarde N is.
```

Als het een False is, hoeft het niet speciaal aan de eindwaarde te liggen. Ook de beginwaarde kan erbuiten liggen, bijvoorbeeld:

```
B = IsBetween(45, 5, 40)      'Is False, want de beginwaarde is hoger.
```

Een wiskundige formule voor grafisch tekenen

Met het Line statement kunnen we een beginpunt en een eindpunt aangeven om lijnen te kunnen tekenen. Willen we lijnen tekenen in een bepaalde richting, dan wordt het andere koek. BASIC heeft geen statements om hulp te kunnen geven voor het makkelijk tekenen van lijnen in een bepaalde richting. De wiskundige formules Sinus en Cosinus moet gebruikt worden om de tekenrichting te bepalen.

Het omrekenen naar X en Y met een opgegeven straal en hoek, gaat met twee formules:

X = S cosinus(H)
Y = S sinus(H)

De horizontale as is altijd de cosinus en de verticale as is altijd de sinus. De sinus van een hoek en de cosinus van de hoek waarmee hij samen 90° vormt (complement) zijn gelijk. Vandaar ook de naam complementaire sinus, dus daarom in BASIC de functienaam Cos.

Onderstaande lus laat zien hoe we kunnen tekenen in een bepaalde hoekrichting:

```
X = U * Cos(W)      'Zie de hoofdstukken: Grafisch programmeren in GW-BASIC
Y = V * Sin(W)
For N = 2 To 360 Step 2
     X1 = N + U * Cos(W)
     Y1 = N + V * Sin(W)
     Line (X, Y)-(X1, Y1)
     X = X1
     Y = Y1
Next N
```

Merk op dat er voor de formules weer het sterretje * ontbreekt. Net als met 5x moet ook hier tussen S en de goniometrische functie vermenigvuldigd worden. In BASIC moeten we dat met het sterretje aangeven, maar in de wiskunde wordt dat niet gedaan.

Werken met tekenreeksen.

We kunnen met waarden werken zoals we het ook opschrijven. Getallen zijn één van de makkelijkste waarden en het lijkt tegenwoordig ook zo gemakkelijk te worden met tekst. Gebruiken we tekst als een waarde, dan gebeurt er eigenlijk heel wat, want een tekst is niet zo maar even een waarde die door een variabele gedragen kan worden.

Tekenreeksen
Een tekst is een tekenreeks (*string*) die uit meerdere tekens bestaat. Dat kan van alles zijn: letters, cijfers en symbolen.
Een tekenreeks wordt niet direct door een variabele gedragen. Het wordt ergens bewaard waar een pointer van de variabele naar verwijst.

In BASIC kan er altijd gemakkelijk met strings gewerkt worden. Wat een C programmeur moet doen om met tekst te kunnen werken, hoeven wij als BASIC gebruikers niet te doen. BASIC neemt al het werk weg, om met een pointer de tekens uit een string te moeten halen. Een voorbeeld is het char statement in C:

```
char Letter;
Letter = 'M';

char *Reeks;
*Reeks = "Hallo Wereld!";

TestFunc(&Reeks);
```

In taal C werkt het erg lastig om met tekst te kunnen werken, maar gelukkig kunnen we in C++ van Visual Studio en Codegear RAD Studio gebruik maken van een string type, net zoals in BASIC en in Basic.

Stringfuncties in Basic
Om met tekst te kunnen werken, moeten we in Basic er wat mee kunnen doen. Hoe lang is (een chinees) de tekst? Hoe halen we een deel uit de tekst? Hoe vervangen we een deel in een tekst? Hoe kunnen we naar delen zoeken en hoe kunnen we een complete string in stukken verdelen?

Houd er rekening mee dat er enkele van de komende functies niet in alle Basic dialecten bekend zijn. Soms bestaan er meerdere functies die hetzelfde uitvoeren. Ik heb er zoveel mogelijk hieronder bijgezet.

Bestaat een functie niet? Jammer, maar er is altijd wel een oplossing om dezelfde functie na te maken. Wat wilt u liever, telkens weer dezelfde code maken of er een functie van maken zodat u het wiel niet nog eens hoeft uit te vinden. In de volgende bulletin gaan we eens aan de slag met stringfuncties en worden bovenstaande vragen beantwoordt.

We kennen bepaalde functies. Hieronder staan ze.

`n = LEN(s)`	Geeft de lengte van string **s** terug. Het gaat om het aantal tekens in de tekenreeks.
`s$ = STR$(n)` `s = CStr(n)`	Converteert de numerieke waarde **n** naar een string en geeft die terug. Controleer welke functie u kunt gebruiken.

`n = VAL(s)`
`n = CInt(s)`
`n = CDbl(s) ...`
	De functie VAL converteert een string naar een numerieke waarde. Is het geen getal, dan is het resultaat een nul. In andere Basic dialecten kunnen we functies gebruiken die de string naar een numeriek type converteert, zoals een Double. De drie punten betekent: er bestaan nog meer converteer typefuncties.

`sd = LEFT$(s, n)` `sd = Left(s, n)`	Geeft een deel van aantal **n** tekens van string **s** terug. Functie LEFT$ geeft de juiste deelstring terug, maar de functie Left geeft bijvoorbeeld in Visual Basic 6 en in VBA een variantwaarde terug. In .NET Basic versies is dat niet meer het geval.
`sd = MID$(s, p, n)` `sd = Mid(s, p, n)`	Geeft een deel van aantal **n** tekens vanaf plaats **p** van string **s** terug. Functie MID$ geeft de juiste deelstring terug, maar de functie Mid geeft in andere Basic dialecten een variantwaarde terug. In .NET Basic versies is dat niet meer het geval.
`sd = RIGHT$(s, n)` `sd = Right(s, n)`	Geeft een deel van aantal **n** tekens van string **s** terug. De start begint aan de rechterkant in de string. Functie RIGHT$ geeft de juiste deelstring terug, maar de functie Right geeft in andere Basic dialecten een variantwaarde terug. In .NET Basic versies is dat niet meer het geval.
`MID$(s1, p, n) = s2` `Mid(s1, p, n) = s2`	Vervangt een deel van aantal **n** tekens vanaf plaats **p** van string **s1** in string **s2**. Is de lengte van **s2** groter dan **n** dan zal alleen het linkerdeel van **s2** met de aantal tekens **n** in **s1** worden vervangen, anders zal gewoon de complete string van **s2** in **s1** worden vervangen. Functie Mid werkt met variantwaarden. In .NET Basic versies is dat niet meer het geval.
`n = INSTR(p, s1, s2)` `n = Instr(p, s1, s2)`	Zoekt de tekenreeks **s2** in **s1** vanaf plaats **p**. De functie geeft de nieuwe plaats terug waar de tekenreeks gevonden is. Geeft een nul terug als de tekenreeks niet gevonden is. Zeer handig voor het ontleden van een complete gegevensstring, zie meer daarover in het volgende Bulletin.

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 NL91 ABNA0495740314
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