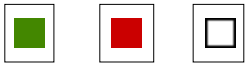
Programmeren in de Wimp

Een cursus programmeren in drie delen voor de RISC OS-omgeving

```
CASE deel% OF
 WHEN 1: PROCvorigboek
 WHEN 2: PROCditboek
 WHEN 2: PROCvolgendboek
ENDCASE
```

Peter Scheele

De uitgever van het RISC OS-magazine ‘Riscosity’ wil een enquête uitvoeren. Daarvoor zijn er twee formulieren in de maak: een papieren en een digitale (een programma met een venster) die met het blad mee naar de abonnee zullen gaan. Op beide formulieren moet de huisstijl van de uitgever worden toegepast en dat zijn een logo, de bedrijfskleur en een lettertype. Voor de papieren versie is dat niet zo moeilijk, maar de programmeur van de digitale versie ziet zich voor enkele problemen geplaatst: het logo als een sprite in het venster en het font in de juiste kleur in het venster met een witte achtergrond zien te krijgen?
Verder wil hij drie radiobuttons (voor ja, nee en geen mening) met eigen ontwerp om de vragen mee te beantwoorden.
En als de muispijl op een van die knopjes staat, moet de pijl in een handje veranderen.



Vraag 1: hoe zet je een sprite in een venster? — sprites in venster
Vraag 2: hoe pas je het lettertype in een andere kleur toe? — tekst in kleur
Vraag 3: hoe verander je de muispijl in een andere vorm? — vorm van de pointer

1.1 een sprite in een venster

Denkstap 1:
Het logo (of andere sprites) moet in een bestand komen te staan. We geven er de naam Sprites (zonder !) aan.
We moeten het logo in de template aanbrengen.
In de initialisatie moet het bestand Sprites geladen worden. Als de template dan geladen wordt, wordt ook de sprite geladen.

Benodigdheden 1:
de sprite van het logo in het bestand Sprites (is als attachment meegestuurd)
de template waarin door middel van een icoon ruimte is aangegeven voor de sprite
een routine waarmee de sprite(s) in bestand Sprites in een lokale spritepool worden geplaatst.

Uitwerking 1:
Maak het bestand Sprites (op de manier die in deel 1 hoofdstuk 3.4 beschreven staat bij !Sprites) en zet daar het logo in.

Open de template in !TemplEd en zet er op de gewenste plaats het icoon in van het tekstbestand (rechts in het ‘Icon palette’). Dubbelklik op dat icoon en het venster verschijnt waarmee het icoon gewijzigd kan worden en stel het volgende in:
Text uit, Sprite aan, Indirected aan, Border aan (voorlopig), Filled uit, Button type Never.
Update & Exit.

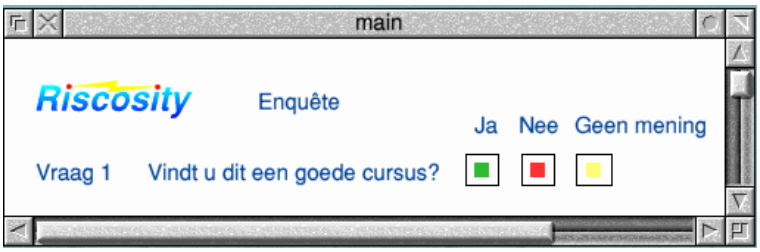
Sleep nu de sprite naar het icoon van !TemplEd op de icoonbalk. Daarmee maak je aan !TemplEd kenbaar dat je niet van de algemene sprite-area gebruik maakt, maar van een lokale. Nu moet de naam van de sprite ingevuld worden in het invoerveld achter Sprite. Dat is niet de bestandsnaam Sprite, maar de naam van de afbeelding, bijvoorbeeld ‘logo’. Je vindt die naam door op Sprite te klikken, de naam staat dan onder de afbeelding. De sprite verschijnt nu in het kadertje in jouw venster. Pas het formaat van dat kadertje aan, zet Border dan uit en tot slot Update & Exit.

Nu kent de template de sprite wel, maar als de template door het programma geladen is, staat de sprite er niet in. Daarvoor moet in de initialisatie (PROCbegin) een routine worden aangebracht, waarmee de sprite(s) geladen worden in een gereserveerd stuk geheugen, de sprite-area. Hoe groot dat stuk moet zijn, weten we eigenlijk niet. Het is van deze toepassing wel te bepalen, maar een algemene routine is misschien wel handig, want die kun je altijd gebruiken. Het idee is: open het bestand “Sprites”, bepaal de grootte ervan met EXT, tel er vier bytes bij voor een header, sluit het bestand, maak een array van de juiste grootte en laad het bestand in het array.

```
file%=OPENIN“<MijnProg$Dir>.Sprites”
size%=EXT#file%+4
CLOSE#file%
DIM sparea% size%
!sparea%=size%:sparea%!8=16
SYS “OS_SpriteOp”,&10A,sparea%,“<MijnProg$Dir>.Sprites”
```

(Bekijk deze SWI eens in de PRM of StrongHelp)

In het bestand met de sprite van het logo zitten drie knopjes voor ja, nee en geen mening. Pas ook die toe in de template.



1.2 tekst in kleur

Denkstap 2:
De kleur van de tekst moet blauw worden. We zouden ook het lettertype kunnen veranderen, maar dat zullen we in een andere case aan de orde stellen. We gaan er nu van uit dat de vragen van de enquête in een icoon van de template komen te staan en we moeten zien of de kleur van de tekst veranderd kan worden in blauw.

Benodigdheden 2:
geen

Uitwerking 2:
Als we in de template een icoon maken, kunnen we bepalen met welke kleur de tekst zal worden afgebeeld. Plaats daarvoor een Data-icoon in de template en dubbelklik daarop. Onderin het venster van ‘Edit icon’ staan Fg col en Bg col voor voorgrond- en achtergrondkleur. Voorgrond wordt blauw, de achtergrond wit. Geen sprite, geen border, niet filled, H centred uit en buttontype never.
Het is het mooist als alle tekst, ook die voor ja, nee en geen mening bij de knopjes, blauw zijn op een witte achtergrond. Het gaat hierbij alleen om de work area, alle andere vensterkleuren blijven ongewijzigd.

1.3 de vorm van de pointer

Denkstap 3:
Het veranderen van de pointer gebeurt door RISC OS op grond van de objecten op de

achtergrond; door vensters of iconen. Bij een icoon kun je aangeven in het veld Validation dat daar een bepaalde pointer geldt. Die pointer moet in het bestand Sprite aanwezig zijn of in de algemene sprite-area. De naam ervan begint met ptr_. Kijk bij StrongHelp bij Wimp, PlotIcon, icon block, icon data, validation strings.

Benodigdheden 3:
Een sprite met de naam spr_naam in de algemene of de lokale sprite-area. We zullen ptr_point gebruiken.

Uitwerking 3:
In het veld Validation van de iconen voor ja, nee en geen mening moeten de volgende strings komen te staan:

Snaam1;Pptr_naam2 oftewel:

Als naam1 gebruiken we ‘ja’, ‘nee’ en ‘gm’. Naam2 is ‘point’.

Sja;Pptr_point, in de volgende Snee;Pptr_point en in de laatste Sgm;Pptr_point
Zet ook Buttontype op radio.

Einde van case 1.

Interactieve help Case 2

Op een basisschool blijven tussen de middag kinderen over. De organisatie daarvan valt niet onder verantwoordelijkheid van de school, maar onder die van de ouders. Die hebben voor de administratie daarvan - het overblijven kost geld - een RISC PC ter beschikking en daarop gaat een programma draaien.
Van alle overblijvers worden gegevens vastgelegd als naam, adres, telefoonnummer, klas, leerkracht. Het programma kan per klas een lijst afdrukken.

De registratie van de overblijvers wordt iedere dag handmatig op lijsten bijgehouden en enkele keren per week kan de data van die lijsten ingevoerd worden in het systeem dat dan de kosten berekent, € 0,60 per overblijf. Het systeem moet vervolgens brieven kunnen afdrukken met rekeningen voor de ouders, binnengekomen bedragen moeten ingevoerd kunnen worden en dan moeten kwitanties worden afgedrukt. Ook moeten er herinneringen of aanmaningen kunnen worden afgedrukt.

De ouders die voor de verwerking zorgen, zijn in het algemeen niet zo deskundig in het computergebruik, vandaar dat de programmeur zich afvraagt hoe de gebruikersvriendelijkheid vergroot kan worden en wil helpberichten (met het programma !Help achter Apps) toepassen. Het gaat om berichten die na een halve seconde verschijnen als de muispijl op een icoon staat.

De vraag:
Hoe zorg je voor helpberichten in RISC OS-toepassingen? (ga zelf op onderzoek uit om te zien welk bestand daarvoor nodig is, wat er in staat en wat de relatie is met iconen in vensters)

De denkstappen:
Deze case heeft betrekking op berichten (messages). Het verkeer dat gestalte moet krijgen heeft te maken met het contact tussen ons programma, het besturingssysteem en het programma !Help. Dat contact verloopt in grote lijnen als volgt: de gebruiker heeft !Help ingeschakeld en beweegt de pointer over de verschillende iconen van het programmavenster waarmee hij bezig is. RISC OS signaleert dat de pointer op icoon #n staat en geeft dat door aan !Help. !Help reageert daarop door op te vragen welk bericht bij dat icoon hoort. Ons programma reageert daarop door het bijbehorende bericht aan !Help door te geven en !Help toont dat bericht. Dat verkeer gaat via de poll-loop door event 17 en 18. We hadden daar de mogelijkheid al aangebracht ons programma via de switcher te beëindigen met de regel:

berichten

!Help

event 17 en 18

```
WHEN 17,18:IF blok%!16=0 THEN klaar%=TRUE
```

Er is dan sprake van berichtenverkeer tussen ons programma en het programma Tasks. Omdat er daar nu meer moet gaan gebeuren, breiden we de mogelijkheden uit door:
`WHEN 17,18:PROContvang` (de uitwerking komt verderop).

De benodigdheden:
We hebben natuurlijk een venster nodig met iconen in de vorm van knoppen, in- en uitvoervelden en sprites. Maar dat hebben we inmiddels.
Verder is er een lijst van berichtregels nodig, waarbij er voor ieder icoon een duidelijke omschrijving is van de werking van dat icoon. De berichtregels kunnen in de vorm van programmaregels in !MijnProg worden opgenomen.
Maar die lijst kan ook in de vorm zijn van een bestand met de naam Messages of

lijst van berichtregels

Berichten dat in de programmadirectory van !MijnProg komt te staan. Bij de start van het programma moet dat bestand geladen worden.We werken nu de eerste mogelijkheid uit, de laatste komt later.

2.1 helpregels in het programma

Er zijn programmaregels nodig om de iconen en de berichten met elkaar te verenigen volgens de specificaties van het berichtenverkeer. Het komt neer op: als de pointer op icoon #n staat, stuur dan bericht #n naar !Help. In programmatermen wordt het iets als:

```
IF icoon%=2 THEN helptekst$=”Hier moet de naam van de
leerling worden ingevoerd.”
```

De uitwerking:
We gaan gebruik maken van SWI Wimp_SendMessage. Die heeft de volgende waarden nodig:
SYS “Wimp_SendMessage”,R0,R1,R2
In R0 komt de reason code uit de Wimp-poll. Dat kan 17 of 18 zijn en we passen 17 toe.
In R1 komt een verwijzing naar het blok.
In R3 komt een handle van het andere programma. We kennen de waarde van die handle niet, maar die staat al in het blok op plaats 4, RISC OS heeft die daar al neergezet. De opdracht wordt nu:
SYS “Wimp_SendMessage”,17,blok%,blok%!4

Wat moet er in het blok:

blok%+0	de grootte van het blok in bytes (max 256)
4	de taskhandle van de zender (!Help, wordt verzorgd door RISC OS)
8	mijn referentie (van de zender, wordt verzorgd door RISC OS)
12	uw referentie (wordt verzorgd door RISC OS)
16	de actiecode (&502 voor helpverzoek en &503 voor helpantwoord)
20	de data (een berichtregel die hoort bij een bepaald icoon)

referenties

Hoe zit dat met mijn en uw referentie? Het is een verwijzingscode naar elkaars berichten. Stel, we hebben partij A en partij B en die communiceren (je ziet iets vergelijkbaars bij correspondentie)
A stuurt naar B en zet bij ‘mijn ref:’ “a” en ‘uw ref’ blijft leeg want die is nog onbekend.
B antwoordt A en vermeldt bij ‘mijn ref:’ “b” en bij ‘uw ref:’ “a”.
Nu reageert A en vermeldt bij ‘mijn ref:’ “a” en bij ‘uw ref:’ “b”.
Er treedt steeds een verwisseling op van de inhoud van ‘mijn ref’ en ‘uw ref’. Zo weet iedere partij waar het over gaat. Die verwisseling moeten we niet vergeten in te bouwen.

actiecodes

En wat zijn die actiecodes?
Codes &502 en &503 geven een bepaald type actie aan. Er zijn er meer:

0	Message_Quit
1	Message_DataSave
2	Message_DataSaveAck
3	Message_DataLoad
4	Message_DataLoadAck
8	Message_PreQuit
&502	Message_HelpRequest
&503	Message_HelpReply

Bijvoorbeeld en een actuele PRM of StrongHelp Wimp geven er minstens tachtig.

De praktijk van bovenstaande:

```
DEF PROContvang
 CASE blok%!16 OF
  WHEN 0:klaar%=TRUE
  WHEN &502:PROChelp
 ENDCASE
ENDPROC

DEF PROChelp
 whandle%=blok%!32:REM Stel vast om welk venster het gaat
 icoon%=blok%!36:REM Stel vast om welk icoon het gaat
 IF whandle%=-2 THEN PROChelptext(”Dit is !MijnProg”)
 IF whandle%=hmain% THEN
  IF icoon%=0 THEN PROChelptext(”Vul in dit venster de voornaam in.”)
  IF icoon%=1 THEN PROChelptext(”In dit venster komt de achternaam.”)
 ENDIF
 IF whandle%=hinfo% THEN
  IF icoon%=0 OR icoon%=1 THEN PROChelptext(”Dit is de naam van het
programma”)
  IF icoon%>1 THEN PROChelptext(”En hier staat de rest van de
informatie.|MLees dit maar rustig door.”):REM |M zorgt voor een nieuwe
regel
 ENDIF
ENDPROC

DEF PROChelptext(t$)
 !blok%=256
 blok%!12=blok%!8:REM Hier zit de verwisseling van ‘mijn ref’ en ‘uw
ref’
 blok%!16=&503:REM Dit is de actiecode voor helpantwoord (HelpReply)
 $(blok%+20)=t$
 SYS “Wimp_SendMessage”,17,blok%,blok%!4
ENDPROC
```

Nog even vier aandachtspunten

Let op!

punt 1

Ten eerste hebben we in deel 1, hoofdstuk 4, § 5 een filter of masker aangelegd om RISC OS duidelijk te maken dat er niet op alle events gereageerd moet worden want dat vertraagt de afhandeling van ons programma. Dat filter moet misschien worden aangepast om event 17 en 18 door te laten. Controleer dat even omdat anders de communicatie via 17 en 18 niet kan plaatsvinden. Beschouw dit als het aanbrengen van een brievenbus.

punt 2

Ten tweede moet er een lijst van berichtcodes worden gemaakt om RISC OS duidelijk te maken welke berichten aan ons programma mogen worden doorgegeven. Beschouw dit als een soort ja/nee-sticker op de brievenbus. We doen dat in PROCbegin door een array aan te maken. Voor iedere berichtcode zijn vier bytes nodig. We gaan er nu drie gebruiken, al zullen het er later wel meer worden.

```
DIM list% 11
```

Nu wordt list% als volgt gevuld:

```
!list%=&502:list%!4=&503:list%!8=0
```

(De 0 aan het eind is om het programma te kunnen stoppen.)

De regel met SYS “Wimp_Initialise”... wordt nu veranderd in:

```
SYS “Wimp_Initialise”,versie,&4B534154,app$,list% TO
,taak%
```

punt 3

Ten derde. Omdat met allerlei berichtregels het programma behoorlijk groter wordt, is het belangrijk in !Run de grootte van WimpSlot te controleren.

punt 4

Ten vierde: de lengte van de berichtregels is van belang. Als zo’n regel uit minder dan twaalf tekens bestaat, beschouwt het besturingssysteem die regel als een code (token) en verwacht een andere manier van afhandeling dan we hierboven hebben toegepast, namelijk de methode van de berichtregels in een bestand, die verderop zal worden uitgelegd. Omdat die afhandeling nu niet is voorzien, loopt het programma vast als we de pointer op het icoon zetten dat dit te korte bericht oproept.

2.2 helpregels vanuit een bestand

Dit was de eenvoudige manier. De moeilijke geeft echter een grotere flexibiliteit omdat er een apart bestand voor gebruikt wordt dat gewoonlijk de naam Messages of Berichten krijgt. Als !MijnProg nu vertaald moet worden in het Engels, dan hoeven de berichtregels (en de Templates) maar vertaald te worden. Maar ieder voordeel heb se nadeel en dat is hier dat deze manier niet geschikt is voor RISC OS-versies 2.

Vijf stappen

We gaan die mogelijkheid toch onderzoeken en dan blijkt al gauw dat er vijf opdrachten voor bestaan. Die komen op het volgende neer:

- Maak een berichtenbestand
- Bepaal de grootte van het berichtenbestand en maak een array aan van die grootte
- Open het bestand
- Sluit het bestand
- Breng icoon en bericht bij elkaar.

1 berichtenbestand

Stap 1: Maak een berichtenbestand
We maken eerst het bestand, gebaseerd op de iconen in het venster van !MijnProg. Jij kent het programma, de werking ervan de functie van de verschillende iconen en je kunt dus zorgen voor concrete beschrijvingen voor ieder icoon. Dat moet wel zorgvuldig, want de toekomstige gebruiker is afhankelijk van de hulp die je gaat geven. Maak dus duidelijke helpteksten.

In verband met de verwerking van onze berichten is het belangrijk aan iedere berichtregel een code toe te voegen. Dan kan ons programma op grond van de pointer op een icoon bepalen welk bericht erbij hoort. Als je meerdere vensters gebruikt, zou je de codering daarop moeten aanpassen als:
V1I2 waarbij V1 staat voor venster 1 en I2 voor icoon 2

De berichtregel wordt dan:
V1I2:Met dit icoon kunt u dit programma afsluiten.

De codes mogen niet langer zijn dan elf tekens, omdat het besturingssysteem dan concludeert dat er geen code staat, maar een tekst en die moet op een andere manier worden afgehandeld.

Dit bestand, van het type Text, wordt in de programmadirectory van !MijnProg geplaatst met de naam Berichten of Messages.

Stap 2: Bepaal de grootte van het berichtenbestand en maak een array aan van die grootte

2 bepaal de grootte

Het bestand moet in de gereserveerde geheugenruimte terecht gaan komen en dat moet daarvoor groot genoeg zijn.
Voor het bepalen van de grootte zijn drie mogelijkheden:
1 de vaste grootte (handmatig in te stellen)
2 grootte bepalen met OPENUP, EXT# en CLOSE
3 grootte bepalen met “MessageTrans_FileInfo”

handmatig

De bepaling van de vaste grootte kan als het bestand klaar is. Zet de pointer op het bestand, klik ‘menu’ en ga via ‘File “Berichten”’ naar Count. Je ziet dan de grootte van het bestand, bijvoorbeeld 384 bytes. Daar moeten vier bytes bij. Maak een array aan met de naam ber_buff% met als grootte 384+4:

```
DIM ber_buff% 388
```

Het werkt uitstekend, maar als het bestand door toevoegingen of vertaling groter wordt, moet handmatig het array ber_buff% vergroot worden. Niet zo handig dus.

met OPENUP

De volgende routine doet het voor ons als we ons programma starten. Het moet opgenomen worden in PROCbegin en het werkt aldus:
- open bestand
  - bepaal grootte
- sluit het bestand
- maak een array van die grootte

```
x%=OPENUP(”<MijnProg$Dir>.Berichten”)
grootte%=EXT#x%
CLOSE#x%
DIM ber_buff% grootte%
```

MessageTrans_ FileInfo

De SWI kent de volgende parameters:
“MessageTrans_FileInfo”,,R1 TO R0,,R2
=> R1 filenaam en die is bij ons: “<MijnProg$Dir>.Berichten”
<= R0 bit is 0 als bericht in het RMA zit of 1 als het in het werkgebied van MijnProg zit
R2 de grootte van de buffer die nodig is.
De uitvoering wordt dus:

```
filenaam$=”<MijnProg$Dir>.Berichten”
SYS “MessageTrans_FileInfo”,,filenaam$ TO ,,grootte%
DIM ber_buff% grootte%
```

En hiermee is er een array aangemaakt dat groot genoeg is om het bestand Berichten of Messages te bevatten.

3 MessageTrans_ OpenFile

Stap 3: We gaan het bestand openen met “MessageTrans_OpenFile”
Deze SWI ziet er als volgt uit:
SYS “MessageTrans_OpenFile”,R0,R1,R2
R0 is een pointer naar een blok (speciaal voor deze toepassing) van vier woorden grootte (128 bytes)
R1 is de bestandsnaam van het berichtenbestand
R2 is een pointer naar de buffer van de berichten

Het blok van R0 moet gemaakt worden door:

```
DIM ber_blok% 128

SYS “MessageTrans_OpenFile”,ber_blok%,filenaam$,ber_buff%
```

In R2 komt filenaam$
R2 bevat de naam van de buffer die de berichten bevat: ber_buff%

4 sluit berichtenbestand

Stap 4: Sluit het berichtenbestand
Dat doen we direct na de OpenFile met de opdracht
SYS “MessageTrans_CloseFile”,R0
In R0 komt de pointer die bij OpenFile ook in R0 stond: ber_blok% dus:

```
SYS “MessageTrans_CloseFile”,ber_blok%
```

5 breng icoon en bericht bij elkaar

Stap 5: Breng icoon en bericht bij elkaar
SYS “MessageTrans_Lookup”,R0,R1,R2,R3,R4,R5,R6,R7 TO ,R1,R2,R3
R0 is de pointer naar het blok voor berichten ber_blok%
R1 is een pointer naar de code van een berichtregel als V1I2
R2 is de pointer naar de buffer ber_buff%
R3 geeft de grootte aan van de buffer met grootte%
R4 is de pointer naar parameter 0 of 0 als die er niet is
R5 is de pointer naar parameter 1 of 0 als die er niet is
R6 is de pointer naar parameter 2 of 0 als die er niet is
R7 is de pointer naar parameter 3 of 0 als die er niet is

R1 is een pointer naar het eindeteken van de code
R2 is de tekststring
R3 is de grootte van de string

We laten R4..R7 buiten beschouwing. De opdracht wordt nu:
SYS “MessageTrans_Lookup”,ber_blok%,code$,ber_buff%,grootte% TO ,t$

t$ Vormt nu de invoer in de SWI SYS “Wimp_SendMessage” Daar kwam de tekst$ in het blok vanaf positie 20 en we doen dat als volgt:

```
$(blok%+20)=t$
```

Samen met nog wat andere zaken die in het eerste stuk al aan de orde waren, hebben we nu genoeg om deze SWI te laten uitvoeren en zal de berichtregel op het scherm verschijnen.

Het programma moet nu als volgt aangepast worden:

In PROCbegin komt te staan:

```
filenaam$=”<MijnProg$Dir>.Berichten”
SYS “MessageTrans_FileInfo”,,filenaam$ TO ,,grootte%
DIM ber_buff% grootte%

DIM ber_blok% 128

SYS “MessageTrans_OpenFile”,ber_blok%,filenaam$,ber_buff%
SYS “MessageTrans_CloseFile”,ber_blok%
```

DEF PROChelp wordt gewijzigd, want daar staan nu niet meer de teksten, maar de codes:

```
DEF PROChelp
 venster%=blok%!32:REM Stel vast om welk venster het gaat
 icoon%=blok%!36:REM Stel vast om welk icoon het gaat
 IF venster%=-2 THEN PROChelptext(”V1-2”)
 IF venster%=hmain% THEN
  IF icoon%=0 THEN PROChelptext(”V1I0”)
  IF icoon%=1 THEN PROChelptext(”V1I1”)
 ENDIF
 IF venster%=hinfo% THEN
  IF icoon%=0 OR icoon%=1 THEN PROChelptext(”V2I12”)
  IF icoon%>1 THEN PROChelptext(”V2I1”):REM |M zorgt voor nieuwe regel
 ENDIF
ENDPROC
```

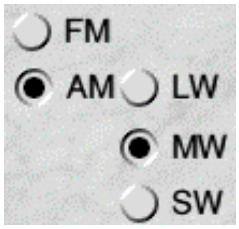
DEF PROChelptext(t$) moet nu ook worden uitgebreid want hierin moeten code en bericht bij elkaar worden gebracht en dan moet het bericht worden getoond:

```
DEF PROChelptext(code$)
 SYS “MessageTrans_Lookup”,ber_blok%,code$,ber_buff%,grootte% TO ,,t$
 !blok%=256
 blok%!12=blok%!8:REM De verwisseling van ‘mijn ref’ en ‘uw ref’
 blok%!16=&503:REM Dit is de actiecode voor helpantwoord (HelpReply)
 $(blok%+20)=t$
 SYS “Wimp_SendMessage”,17,blok%,blok%!4
ENDPROC
```

Hier eindigt case 2.

Het testen van de instelling van radiobuttons

Case 3



Iemand is bezig met het ontwerpen van een programma. Hij heeft er al een zeker idee van voor ogen en hij realiseert zich dat het handig is om in het venster enkele radiobuttons te gebruiken. Terwijl hij de template maakt, vraagt hij zich af hoe de stand van die buttons moet worden gecontroleerd. Immers, de gebruiker klikt een button aan en nu moet het programma op die nieuwe instelling gaan reageren.

De vraag
Hoe kun je het programma laten vaststellen welke stand radiobuttons hebben?

De denkstappen
Radiobuttons zijn knopjes met stand ‘aan’ of ‘uit’. Denk aan buttons met het label ‘Option’ en ‘Choice’. Ze kunnen afzonderlijk voorkomen, maar ook als groepen, waarbij het kan zijn dat er in zo’n groep een knop aan kan staan terwijl de rest uit staat. We noemen zo’n groep ESG ofwel Exclusive Selection Group. We kunnen in de template aangeven, per icoon, dat ze het button type ‘Radio’ moeten hebben en door het ESG-nummer tot welke groep ze behoren. In de afbeelding horen FM en AM bij ESG 1 en LW..SW bij ESG 2.

Option
Choice

Om de stand van een icoon vast te stellen, is er de SWI SYS “Wimp_GetIconState”

Want ieder icoon heeft vlaggen of bits en daarvan moeten we die zien te vinden die de stand van het icoon bevat. Als we het bit gevonden hebben, moeten we nagaan of dat bit op nul of op een staat. Vervolgens kunnen we het programma op grond van de uitkomst actie laten ondernemen.

De benodigdheden
Een venster met een of meer radiobuttons.
SYS “Wimp_GetIconState”,,R1
waarbij R1 een pointer is naar een blok. In dat blok moet op plaats 0 de window handle en op plaats 4 de icon handle. Na uitvoering staat vanaf plaats 8 een iconblock dat bestaat uit 32 bytes. Ergens in dat stuk data is er een plaats, een bit, dat de stand van het icoon aangeeft. Het is zaak dat bit op te zoeken, de stand ervan te bepalen met AND en een masker zodat er verdere actie kan plaatsvinden.

De uitwerking

3.1 met één radiobutton

Als er maar een radiobutton in het venster staat, kunnen we volstaan met de volgende manier:
In PROCbegin maken we een variabele stand% en die krijgt de waarde van het button: aan of TRUE, uit of FALSE: `stand%=TRUE`
en we maken daar een variabele knop% en die geven we de waarde stand%:
`knop%=stand%`

TRUE en FALSE

we laten bij PROCmuis testen op

```
IF (icoon%=4) AND (klik%=4) THEN knop%=NOT(stand%)
```

Daarmee verandert de waarde van TRUE in FALSE. Als er weer op het knopje wordt geklikt, keert de waarde weer op in TRUE. (TRUE heeft overigens het getal -1 en FALSE het getal 0 en een bit kan de waarde 0 of 1 bevatten.)

TRUE=-1
FALSE=0

Tot slot kunnen we met de waarde in knop% iets doen door:
IF knop%=TRUE THEN actie-1 ELSE actie-2

3.2 met enkele radiobuttons

Als je meerdere buttons gebruikt, wordt bovenstaande manier wat omslachtig. Er is daarvoor een efficiëntere methode. We gaan even uit van drie knoppen voor ‘ja’, ‘nee’ en ‘geen mening’. We verbinden daar de variabelen ja%, nee% en geenm% aan. De iconen van de knoppen hebben de nummers 6, 7 en 8.
We kunnen nu met “Wimp_GetIconState” voor icoon 6 de instelling opvragen met:

```
!blok%=hmain%:blok%!4=6
SYS “Wimp_GetIconState”,,blok%
```

Nu staat op plaats 24 in het blok 12 bytes aan icoondata. In die 12 bytes staat op bit 21 (Icon is (already) selected of Icon is selected by user, and is inverted) de instelling van icoon 6. Die instelling moeten we hebben. We weten dat die instelling een nul of een een kan zijn, maar we weten niet welke daarvan actueel is.

masker

De truuk om dat bit zichtbaar te maken, is het te vergelijken met een bit waarvan we de waarde wel kennen, een één bijvoorbeeld. We noemen dat een masker. De vergelijking ziet eruit als:

logische AND

bit AND masker = waarde waarbij waarde 1 kan zijn of 0.
Nu kan het bit een van twee waarden hebben (0 of 1) en het masker kan ook een van twee waarden hebben (ook 0 of 1). Daardoor zijn er vier vergelijkingen mogelijk:
1 AND 1 = 1
1 AND 0 = 0
0 AND 1 = 0
0 AND 0 = 0

Omdat we duidelijkheid willen, nemen we een masker met de waarde 1. Er blijven dan twee vergelijkingen over:
1 AND 1 = 1
0 AND 1 = 0
Of anders gezegd: als de vergelijking 1 oplevert, weten we dat bit 1 was:
? AND 1 = 1
Als we nu een byte hebben (8 bits) en we willen weten wat de stand is van bit 4 dan moeten we een masker maken dat bit 4 eruit filtert.

```
byte     ????????
AND
masker   00010000
=        00010000 geeft aan dat bit 4 in de byte op 1 staat.
```

We mogen de voorloopnullen weglaten waardoor het masker 1000 wordt. Maar we zien ook dat de uitkomst groter is dan 0, want het is 16.

In programmeertermen mogen we het zo doen:

```
masker%=%1000
IF byte% AND masker%>0 THEN ...
```

Als het geen byte is maar het zijn er 12 en we willen weten wat de stand is van bit 21 dan doen we dat zo:
((blok%!24) AND %10000000000000000000) of korter
((blok%!24) AND &200000) (zoek nog even in deel 1, hoofdstuk 3.6 op hoe je van binair naar hex converteert).

```
IF ((blok%!24) AND &200000)>0 THEN ...
```

De bovenstaande bewering is eigenlijk niet nauwkeurig genoeg. Die uitkomst is namelijk niet 1. Het kan 0 zijn, of een waarde die (veel) groter is dan 0.
Wat eraan gedaan moet worden is dat er een verschuiving moet gaan plaatsvinden.

Het bit waarom het gaat moet helemaal naar rechts worden geschoven en als masker testen we dan niet met &200000 maar met 1. Dat schuiven doen we met een Shift-right (>>) en wel 21 posities. shift right
We schrijven dat als blok%!24>>21. De vergelijking (of de test) wordt nu:
(blok%!24>>21) AND 1

In ons programma wordt het dus:
```
!blok%=hmain%:blok%!4=6
SYS “Wimp_GetIconState”,,blok%
IF ((blok%!24>>21) AND 1)=1 THEN ...
```

Nu weten we dat van icoon 6, maar we willen dat ook bepalen van de andere twee. Het is wel handig daar een functie voor te maken als:
```
ja%=FNicoon(6)
nee%=FNicoon(7)
geenm%=FNicoon(8)

DEF FNicoon(ic%)
!blok%=hmain%:blok%!4=ic%
SYS “Wimp_GetIconState”,,blok%
=((blok%!24>>21) AND 1)
```

ja%, nee% en geenm% onderscheiden zich nu van elkaar omdat twee ervan een 0 bevatten en een ervan een 1. Daar kunnen we verder mee als:
```
IF ja%=1 THEN actie1
IF nee%=1 THEN actie2
IF geenm%=1 THEN actie3
```

3.3 met meerdere radiobuttons

Bij nog meer radiobuttons wordt ook dit wel wat omslachtig. Er is dan ook een derde manier waarbij we van een groep iconen kunnen bepalen bij welke iconen bit 21 op 1 staat. We hebben daar SWI “Wimp_WhichIcon” voor nodig met de volgende parameters:
SYS “Wimp_WhichIcon”,R0,R1,R2,R3 TO ,R1
waabij:
R0 de windowhandle bevat (hmain% bijvoorbeeld)
R1 wijst naar een blok waarin de handles (of icoonnummers) staan van de iconen die getest moeten worden
R2 het masker bevat (&200000)
R3 een specificatie bevat van de bits die getest moeten worden (in onze opzet bit 21 of &200000)

R1 bevat na uitvoering een lijst van van de handles (icoonnummers) van iconen die voldoen aan de test.

We hebben er een buffer voor nodig dat bestaat uit een aantal bytes dat evengroot is als het aantal iconen dat we willen testen en die maken we in PROCbegin:
De programmaregels worden dan:

```
DIM iconblok% 3
!iconblok%=6:iconblok%!1=7:iconblok%!2=8
SYS “Wimp_WhichIcon”,hmain%,iconblok%,&200000,&200000 TO ,iconblok%
```

Bij een ESG zit er in iconblok% nu maar één waarde en dat is die van het icoon dat aan staat, 6 bijvoorbeeld.

Tot zover Case 3

Laden en bewaren, drag and drop Case 4

Je hebt het voor elkaar. Je hebt een pracht van een spelletje gemaakt en het werkt. Je begon meteen te spelen. De eerste levels gingen nog wat traag, want je was er nog niet zo bedreven in, maar nu vliegen je vingers over het toetsenbord. De score wordt snel hoger, de uitdaging groeit met de minuut, je schuift steeds meer naar het puntje van je stoel. Zelfs die hindernis die eerst zo moeilijk leek, blijkt vliegensvlug te nemen. Je wordt je ervan bewust dat het gaat lukken: het laatste level komt binnen handbereik. Je wringt je in allerlei bochten om het sturen zo goed mogelijk te doen en jawel... het is zover, je treedt toe tot de hall of fame.
Had je nu maar een highscorebestand ingebouwd. Maar hoe laad en bewaar je een dergelijk bestand?

De vraag:
Hoe laad en bewaar je bestanden?

De denkstappen:
Er zijn verschillende laad- en bewaaracties mogelijk die ook op verschillende wijzen uitgewerkt moeten worden om ze toepasbaar te maken:

1 het laden en bewaren zonder actie van de gebruiker. Het programma laad het bestand tijdens het starten en als de data veranderd is, bewaart het programma die automatisch. Het bestand staat in de programmadirectory. Denk aan de highscore van bovenstaand voorbeeld.
2 het laden en bewaren worden wel door een actie van de gebruiker uitgevoerd, maar alleen indirect en het bestand staat alleen maar in de programmadirectory. Stel je een configuratiebestand voor dat geopend wordt door een klik op menukeuze of knopje ‘Instellingen’ en dat, na wijzigingen op het scherm met het knopje ‘Save’ weer bewaard wordt.
3 het laden en bewaren worden door actie van de gebruiker uitgevoerd. Het bestand kan nu overal staan. De gebruiker klikt op het bestandsicoon of sleept het naar het programma-icoon op de balk of in het venster van het programma. Nu wordt het bestand geladen. Het is de welbekende wijze onder RISC OS en er hoort ‘drag-and-drop’ bij.
Volgens hetzelfde bekende principe kan het bestand bewaard worden waar de gebruiker het maar hebben wil, door het daarnaar toe te slepen.

4.1 bestandsformaten

Voordat we bovenstaande concepten gaan uitwerken is het van belang een keuze te maken. We moeten namelijk bepalen welk bestandsformaat ons programma zal kunnen verwerken. Er zijn twee mogelijkheden:
- een van de bestaande formaten of
- een formaat naar eigen keuze.

Een bestand heeft altijd een type of formaataanduiding. Je kunt het zichtbaar maken door de muispijl op een bestand te zetten en op menu te klikken. Ga dan via de bestandsnaam naar Info en in een venster is nu het Type zichtbaar. Dat type kan op twee manieren worden weergegeven: als hexcode of als naam. Voor de hexcode worden drie tekens gebruikt. Er zijn onder RISC OS in totaal &FFF of 4096 verschillende bestandsformaten mogelijk. Een aantal daarvan is in gebruik voor standaardtoepassingen. Bijvoorbeeld:

Text	&FFF	Sprite	&FF9	Desktop	&FEA
Basic	&FFB	Obey	&FEB	PrntDefn	&FC6

Het gebied van &E00 tot &FFF (512) is daarvoor gereserveerd en de bestanden mogen niet voor andere doeleinden gebruikt worden dan waar ze voor bedoeld zijn.

Maar ook algemene toepassingen als ArtWorks, OHP, StrongHelp of Ovation kennen hun eigen bestandsformaten:

OvnPro	&B27	ArtWorks	&D94
StrongHelp	&3D6	OHP	&AC1

Het gebied tussen &800 en &DFF (1536) is gereserveerd voor softwarebedrijven. Ook deze formaten mogen niet voor iets anders gebruikt worden.

In de PRM is een hoofdstuk gewijd aan gangbare bestandsformaten en daarin staan specificaties waaraan diverse bestandstypes moeten voldoen. Het voert te ver om er hier op in te gaan, maar je vindt de informatie in hoofdstuk ‘File formats’; je komt er beschrijvingen tegen van Sprite-, Template-, Draw-, Font- en Musicfiles. Het betekent dat jouw programma bestanden moet kunnen maken volgens de beschreven specificaties. Dan kunnen die bestanden ook in andere programma’s worden opgevraagd. Maakt jouw programma een Sprite-bestand, dan moet dat bestand ook met !Paint geopend kunnen worden.

Het overige deel, van &000 tot &7FF, en dat zijn er 2048, is beschikbaar voor gebruikers en daarvan mag je er enkele reserveren. Je loopt er echter het risico mee dat een collega-programmeur die ook gekozen heeft. Wie dan op een bestand klikt met de bedoeling jouw programma te laden, ziet dan dat er een ander programma geladen wordt. Kies je formaat dus wat zorgvuldig.

De nieuwere PRM’s geven een iets andere indeling aan:
E00 - FFF allocated by Acorn for generic data types
B00 - DFF allocated by Acorn to software houses for applications
A00 - Aff reserved for use by Acorn applications
400 - 9FF allocated by Acorn to software houses for applications
100 - 3FF allocated by Acorn to public domain software
000 - 0FF free for users
en dan blijven er voor gebruikers ‘maar’ 256 tot 1024 mogelijkheden over.

Je mag er een eigen sprite aan toekennen. Dat krijgt de naam file_XXX waarbij op de plaats van de ixen de hexcode van jouw bestandstype komt te staan. Kijk nog even in deel 1 hoofdstuk 3.4 een sprite-file hoe dat moet. De sprite met de naam file_XXX komt in het bestand !Sprites te staan. Bedenk dat een bestand een kader om de sprite krijgt.

Als je een bestand bewaart op de wijze van:
- open bestand
- bewaar variabelen
- sluit bestand
wordt het bestandstype automatisch op Data (&FFD) gezet. Om dat te veranderen, neem je in je programma, na het sluiten van het bestand, de volgende regel op:

```
OSCLI “SetType “+bestnaam$+” XXX”
```

waarbij bestnaam$ de bestandsnaam bevat en de ixen staan voor de hexadecimale aanduiding.

We gaan de denkstappen nu verder uitwerken.

4.2 automatisch laden en bewaren

Denkstap 1:
Het laden en bewaren zonder actie van de gebruiker. Het programma laad het bestand tijdens het starten en als de data veranderd is, bewaart het programma die automatisch. Het bestand staat in de programmadirectory. Denk aan de highscore van bovenstaand voorbeeld.

Overweeg nog even het volgende. Als jij een nieuwe versie hebt gemaakt van jouw programma, dan zal een gebruiker de nieuwe versie kopiëren over de oude. Daarmee raakt hij zijn highscores kwijt. Om dat te voorkomen, wordt aangeraden dit soort bestanden (denk ook aan configuratiebestanden) buiten de programmadirectory te bewaren. En een geschikte plaats is in Boot.Choices. Daar moet nu een map staan met de naam van het programma. Het pad is <Choices$Write>.MijnProg

Hoe komt die map daar? Je kunt in PROCbegin een test bouwen. Die controleert of de map MijnProg al bestaat in Boot.Choices. Als die er niet is, komt er een 0 in f% te staan en dan start je de procedure eerstekeer.

```
f%=OPENIN(”<Choices$Write>.MijnProg.HighScores”)
IF f%=0 THEN CLOSE#f%:PROCeerstekeer ELSE CLOSE#f%

DEF PROCeerstekeer
 *CDir <Choices$Write>MijnProg
 REM Maak daar nu het bestand of kopieer het
 REM vanuit de programmadirectory daar naartoe
ENDPROC
```

Benodigdheden 1:
Een highscorebestand in de directory van !MijnProg.
En een venster met een aantal regels voor de hoogste scores.

Uitwerking 1:
Tijdens het starten wordt de data van het bestand geladen op de volgende wijze:
- maak array’s voor de variabelen
- open bestand
- lees variabelen
- sluit bestand

Of in programmatermen:

```
DIM naam$ 9,score% 9
x%=OPENUP(”<Choices$Write>.MijnProg.HighScore”)
 FOR I%=0 TO 9
  INPUT #x%,naam$(I%),score%(I%)
 NEXT I%
CLOSE #x%
```

Dit gebeurt in PROCbegin, want het hoeft maar eenmalig. Het valt buiten het bestek van de cursus om te behandelen hoe de nieuwe data moet worden bewerkt. Er moet natuurlijk beoordeeld worden of de nieuw behaalde score een plaats verdient in de highscores en op welke positie.

Als de data dusdanig wijzigt dat het bestand bewaard moet worden, gaat dat op de volgende wijze:
- open bestand
- schrijf variabelen
- sluit bestand

Het bestand moet eenmalig worden aangemaakt. Dat kan door een routine te maken en die in PROCeerstekeer te zetten.

```
DIM naam$ 9,score% 9
x%=OPENUP(”<Choices$Write>.MijnProg.HighScore”)
 FOR I%=0 TO 9
  naam$(I%)=””:score%(I%)=0
  PRINT #x%,naam$(I%),score%(I%)
 NEXT I%
CLOSE #x%
```

Het maken van het venster voor de highscores en het plaatsen van de data in de iconen van het venster staat beschreven in deel 1 paragraaf 5.3 en 6.2.

4.3 laden en bewaren door actie van de gebruiker

Denkstap 2:
Het laden en bewaren worden wel door een actie van de gebruiker uitgevoerd, maar alleen indirect en het bestand staat alleen maar in <Choices$Write>. Met indirect bedoel ik dat de gebruiker niet direct opdracht geeft om het bestand te laden, maar dat het laden voortvloeit uit een andere actie. Stel je een configuratiebestand voor dat geopend wordt door een klik op menukeuze of knopje ‘Instellingen’ en dat, na wijzigingen op het scherm met het knopje ‘Save’ weer bewaard wordt.

Benodigdheden 2:
Een configuratiebestand in de directory Boot.Choices
Een menukeuze of een knopje ‘Instellingen’
Een venster om de instellingen te tonen en te kunnen wijzigen met een knopje ‘Save’ om de wijzigingen op te slaan en een knopje ‘Ok’ om de wijzigingen meteen actief te maken.

Uitwerking 2:
Ook hier moet er een bestand worden gemaakt waarin de instellingen komen te staan. Dat kan met bovenstaand programmadeel, al zullen er andere variabelen en waarden worden bewaard. Het laden en bewaren gebeurt ook op vergelijkbare wijze.

Het maken van menu’s en het activeren van keuzes staat al beschreven in deel 1, paragraaf 7.2.

Het maken van een venster met knoppen en het afhandelen van acties staat beschreven in deel 1 paragraaf 5.3 en 6.1.

Omdat de data die bewaard moet worden niet zo’n duidelijke structuur heeft als de highscoretabel, kan de manier van bewaren ook anders zijn. We gebruiken er BPUT en BGET voor.
- open bestand
- bewaar de data met BPUT variabele of lees die met BGET variabele
- sluit bestand

Er is geen lus voor nodig. omdat er geen sprake is van herhaling van datapatronen.

4.4 laden en bewaren met drag and drop: eerst bewaren

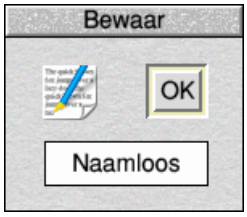
Denkstap 3:
Het laden en bewaren worden door actie van de gebruiker uitgevoerd. Het bestand kan nu overal staan. De gebruiker klikt op het bestandsicoon of sleept het naar het programma-icoon op de balk of in het venster van het programma. Nu wordt het bestand geladen. Het is de welbekende wijze onder RISC OS en er hoort ‘drag-and-drop’ bij. Volgens hetzelfde principe kan het bestand bewaard worden waar de gebruiker het maar hebben wil, door het daar naar toe te slepen.
Dit is het meest uitgebreide, maar ook het meest interessante deel. Laten we eens uiteenrafelen wat er moet gebeuren.

Het bewaren wordt gestart door op menukeuze of knopje ‘Bewaar’ te klikken. Er verschijnt nu een venstertje met een icoon, een invoerveld en een Ok-knopje. Door het icoon naar de plaats van bestemming te slepen, wordt het bewaard. Als pad en naam van het bestand eenmaal bekend zijn, mag op Ok geklikt worden. Als op Ok wordt geklikt en pad en naam zijn niet bekend, moet een melding verschijnen.
We kijken eerst naar het bewaren en verderop naar het laden.

Benodigdheden 3:
Een bewaarvenster of savebox,
drag and drop

Uitwerking 3:
We gaan met !TemplEd een bewaarvenster of savebox maken. Start !TemplEd en open het templatebestand van !MijnProg. Maak daar een nieuw venster met als naam ‘Save’. Het bevat:
- een titelbalk met daarin de titel Bewaar of Save.
- een sprite
- een invoervenster
- een Ok-knop.

Er zijn geen andere vensterattributen als [X], [-] of scrollbars.



Kies een passende bestandssprite uit ROMSprites (kijk bij deel 1 paragraaf 3.4 hoe je die ROMSprites zichtbaar maakt) en voer de naam ervan in in het invoerveld Sprite van de template.

Als je een eigen sprite wilt maken, kijk dan bij deel 2, case 1 hoe het logo van de uitgeverij in de template werd geplaatst. Maak de sprite met als naam file_XXX, waarbij XXX de hexcode van het bestandsformaat voorstelt. In case 1 staat ook beschreven hoe je de sprite in de spritepool zet.

In alle gevallen moet de sprite dragable worden gemaakt.

Het invoervenster maak je zo dat er bij het invoericoon alvast een bestandsnaam staat. Bijvoorbeeld ‘Naamloos’. De maxsize moet op 256 staan in verband met de lange padnaam.

Het is ook nodig de validation op “A~ “ te zetten, A tilde spatie. Dat voorkomt dat er spaties in de bestandsnaam worden ingevoerd.

Zet er een klein Ok-knopje bij.

Vraag de statistics op van deze template en bewaar het.

Zorg nu dat in PROCbegin ook dit venster aangeroepen wordt, maar open het daar nog niet. Kijk bij deel 1 paragraaf 5.4 hoe dat moet. Geef de handle de naam hsave%.

Bereid het menu uit met een keuze Bewaar en laat daarmee de savebox openen. Het gaat op vergelijkbare wijze waarop, vanuit het menu, de infobox geopend werd. Kijk bij deel 1 hoofdstuk 7.2.

Zet in het hoofdvenster een knopje 'Bewaar' en zorg, via PROCmuis, dat na een klik op dat knopje de savebox geopend wordt.

Er is nog een plaats waar het bewaren gestart kan worden. Als je het programma stopt, moet onbewaarde data nog bewaard worden. Dus vlak voor klaar%=TRUE moet de bewaarprocedure worden uitgevoerd.

Als dat gelukt is, wordt het tijd de drag and drop te bespreken.

Ook voor het bewaren en laden door middel van drag and drop hebben we de SWI "Wimp_SendMessage" nodig. Die hebben we in Case 1 al gezien. Er zijn vier parameters R0, R1, R2 en R3. De laatste drie worden verderop duidelijk. In R0 moet de reasoncode vermeld worden uit de Wimp-poll. We hebben in de poll-lus event 17 en 18 bij elkaar genomen met:

```
WHEN 17,18:PROContvang
```

In Case 1 gebruikten we reasoncode 17. Wat doet 18 en wat moeten we met 19?

We gebruiken 17 en 18 als !MijnProg moet communiceren met andere taken. Bij sommige communicaties is het van groot belang dat de andere taak reageert en bij andere communicaties is het toegestaan dat de andere taak niet reageert.
Als ik een bestand wil bewaren, *moet* de Filer reageren door middel van een bevestiging (Ack, van acknowledge). Als dat niet gebeurt, moet er een melding komen om de gebruiker te verwittigen dat het bewaren niet lukt. Als ik daarentegen helpberichten wil laten zien (zoals in Case 2) en !Help is niet ingeschakeld en niet kan reageren, wil ik daar geen melding van. Stel je voor, dan verschijnt er steeds een melding als ik de pointer te lang op een icoon laat staan.
Kortom: we gebruiken event 17 voor communicatie zonder bevestiging en 18 voor de bevestiging, waarbij bijzonderheden via 19 gemeld worden.

Bij Case 2 hebben we al gezien dat we in PROContvang kunnen testen op verschillende soorten berichten:

```
DEF PROContvang
 CASE blok%!16 OF
  WHEN 0:klaar%=TRUE
  WHEN &502:PROChelp
 ENDCASE
ENDPROC
```

(Kijk in Case 2 vooral punt 1 t/m 4 nog even na.)

Nu komen daar voor het bewaren en laden nog de volgende berichten bij:

nummer	naam	reasoncode
1	Message_DataSave	(17)
2	Message_DataSaveAck	(18)
3	Message_DataLoad	(17)
4	Message_DataLoadAck	(18)
6	Message_RAMFetch	
7	Message_RAMTransmit	

Even terzijde
Laten we eerst even zorgen voor list% (je weet wel, de ja/nee-sticker) die we in Case 2 bij de interactieve Help-berichten al hebben gezien. Die moet worden uitgebreid. We deden dat in PROCbegin door een array aan te maken. Voor iedere berichtcode zijn vier bytes nodig. We gaan er nu zes gebruiken.
```
DIM list% 23
```

Nu wordt list% als volgt gevuld:
```
!list%=1:list%!4=2:list%!8=3:list%!12=&502:list%!16=&503
:list%20=0
```
(De 0 aan het eind is om het programma te kunnen stoppen.)

Omdat we steeds van de bevestiging gebruik willen maken (het bewaren en laden moet zo veilig mogelijk gebeuren), kiezen we voor DataSave en DataSaveAck. Deze keuze bepaalt het verloop van de communicatie. We breiden PROContvang uit met:
```
WHEN 2:PROCdatasaveack
WHEN 3:PROCdataload
```
(bekijken we later)

Hoe kunnen we er nu voor zorgen dat via poll-event 17,18 een bericht binnenkomt zodat PROCbewaar geactiveerd gaat worden? Het gaat om slepen; de volgende stappen laten zien wat er gebeurt (we gaan er vanuit dat de gebruiker het bewaarvenster opent en het bewaaricoon naar een bestemming sleept). De volgorde volgens de PRM is:

1 de gebruiker beëindigt de drag met het bewaaricoon, onze applicatie ontvangt een User_Drag_Box event
2 onze applicatie roept Wimp_GetPointerInfo aan om schermcoördinaten en venster/icoonhandles vast te stellen op het einde van de drag (de drop).
3 onze applicatie stuurt een DataSave (nummer 1 via reasoncode 17) met het padnaam van het bestand naar de Filer
4 de Filer antwoordt met DataSaveAck(2 via 18) dat de complete padnaam bevat
5 onze applicatie bewaart de data
6 onze applicatie stuurt een DataLoad-bericht(3 via 17) naar de Filer
7 de Filer antwoordt met DataLoadAck(4 via 18)

De laatste twee zijn vanwege compatibiliteit met vergelijkbare protocollen.
Je ziet dat vanuit onze toepassing gezien het bewaren steeds gebeurt via poll-event 17 en dat de bevestiging steeds via 18 terugkomt.

De manier waarop wij het zien is meer als:

- de gebruiker opent het bewaarvenster en sleept het bewaaricoon naar zijn bestemming.
1 de gebruiker beëindigt de drag met een drop. Die wordt via poll-event 7 gemeld. Wij starten daar PROCdrop.
2 in PROCdrop wordt Wimp_GetPointerInfo geactiveerd.
3 De DataSave wordt gestart in PROCdrop. Daarin moet de padnaam klaargemaakt en doorgegeven worden aan de Filer via Wimp_SendMessage 17,1.
4 ons programma ontvangt via PROContvang een DataSaveAck, via 2, met de complete padnaam. Wij maken PROCdatasaveack en zoeken de padnaam.
5 ons programma bewaart de data op de aangegeven plaats door middel van PROCbewaar.
6 ons programma verstuurt een DataLoad via 3 naar de Filer. Wij doen dat met PROCdatasaveack.
7 de Filer antwoordt nog met een DataLoadAck via 4, maar daar hoeven wij niet op te reageren.

Hoe werkt dat uit? (zie ook het overzicht aan het eind van de case)
Het slepen vindt zijn oorsprong in PROCmuis. Daar wordt gedetecteerd waar de muis staat en op welke knop gedrukt wordt. Nu levert een klik waarde 4, maar de linkermuisknop ingedrukt houden levert de waarde 64. De regel
`IF venster%=hsave% AND icoon%=0 AND klik%=64 THEN PROCdrag(hsave%,0)` doet het werk.

Het slepen heeft een begin, maar ook een eind. Het wordt door het besturingssysteem via de poll-loop event 7 doorgegeven. Het einde van de drag wordt bepaald in PROCdrop door middel van “Wimp_GetPointerInfo” en nu kan het pad bepaald worden.
Dan vindt verzoek aan de Filer plaats om een bestand te bewaren door middel van “Wimp_SendMessage”, 17, 1.

Als de communicatie met de Filer succesvol is, komt het berichtenverkeer op gang via event 17,18 PROContvang, WHEN 2:PROCdatasaveack waarin nu via “Wimp_SendMessage” het bestand bewaard zal worden. We doen dat met PROCbewaar.

Dan moet een DataLoad via 3 naar de Filer gestuurd worden. Dat doen we ook in PROCdatasaveack.
Tot slot moet het bewaarvenster gesloten worden. Dat doen we in PROCbewaar.

Als er in dat verkeer iets mis gaat, wordt dat via event 19 gemeld. Het bericht dat daar binnenkomt, moet door PROCmelding(bericht$) (deel 1, hoofdstuk 8.7) worden weergegeven.
Het gaat in PROChoofd, de poll-loop, met:
```
WHEN 19:PROCmelding(”Bewaren mislukt, probeer het nog eens”,1)
```

De uitwerking van de procedures:
PROCdrag werkt met de SWI “Wimp_DragBox” en die vraagt als R1 een pointer naar een blok. Dat blok bevat de data van het bewaarscherm en is het blok van hsave%.
Het blok moet verder de volgende data bevatten:
het type drag, de coördinaten van de dragbox (het kleine vierkantje van het bewaaricoon) bij het begin van de drag en de coördinaten van de bounding box (het gehele scherm).

blok%+4	drag type
8	min. x dragbox
12	min. y dragbox
16	max. x dragbox
20	max. y dragbox
24	min. x bounding box (=0)
28	min. y bounding box (=0)
32	max. x bounding box (heel groot)
36	max. y bounding box (=ook heel groot)

Er zijn verschillende soorten drag-types. Je kunt ze vinden in de PRM of bij StrongHelp bij Wimp_DragBox. We kiezen hier voor type 5 en dat laat tijdens het slepen een rechthoek zien.

Het heeft geen zin bovenstaande data er met de hand in te zetten, want met “Wimp_GetWindowState” en “Wimp_GetIconState” krijgen we die waarden en we hoeven ze maar door te geven; de volgende procedure doet het voor ons:

```
DEF PROCdrag(venster%,icoon%)
 $padnaam%=""
 !blok%=venster%:REM het venster van waar gesleept wordt
 SYS "Wimp_GetWindowState",,blok%
 x%=blok%!4-blok%!20
 y%=blok%!16-blok%!24
 blok%!4=icoon%:REM icoonnummer van waar gesleept wordt
 SYS "Wimp_GetIconState",,blok%
 blok%!4=5:REM Hier staat de drag type, we kiezen voor
type 5, maar de PRM geeft er meer
 blok%!8=x%+blok%!8:REM x- en y-waarden van de dragbox
 blok%!12=y%+blok%!12
 blok%!16=x%+blok%!16
 blok%!20=y%+blok%!20
 blok%!24=0:REM x- en y-waarden van het scherm
 blok%!28=0
 blok%!32=&7FFFFFFF:REM grote waarde, past altijd
 blok%!36=&7FFFFFFF
 SYS "Wimp_DragBox",,blok%
ENDPROC
```

Dit is overigens een standaardprocedure die zonder aanpassing in al deze situaties gebruikt kan worden.

Nu melden we het bewaren voor het eerst aan bij de Filer. Dat doen we met PROCdrop. Die gebruikt "Wimp_SendMessage" en die vraagt de volgende parameters:
R0 de reason code en dat is de code van de communicatie: 17
R1 berichtenblok
R2 windowhandle van bestemming (zie !20)
R3 icoonhandle van de bestemming (zie !24)

Het berichtenblok voor "Wimp_SendMessage" moet de volgende data gaan bevatten:

blok%+0	grootte van het blok
4	taskhandle van zender
8	mijn referentie
12	uw referentie (Filer)
16	de code voor Message_DataSave
20	window handle van bestemming
24	icoonhandle van bestemming
28	schermpositie x
32	schermpositie y
36	geschatte bestandsgrootte of -1 als het bewaren mislukt is
40	bestandstype
44	het hele pad

"Wimp_GetPointerInfo" kan nagaan waar op het scherm de pointer zich bevindt en kan ook handles doorgeven van vensters of iconen. We nemen die data aan en zetten dat op positie 20 t/m 32 in een blok, dat we meteen kunnen gebruiken.

De geschatte bestandsgrootte kennen we niet, maar is, in een aantal gevallen, met LEN variabele te bepalen. Ook kun je misschien de omvang bepalen van de gevulde array's die je wilt bewaren. Het hoeft maar een schatting te zijn.

De inhoud van blok%!44 wordt het hele pad, inclusief bestandsnaam en daarvoor moeten we een array aanmaken in PROCbegin, met als naam padnaam% en grootte 256:

```
DIM padnaam% 256
```

We vullen het array meteen met de defaultnaam ‘Naamloos’ door:

```
$padnaam%=”Naamloos”
```

Als de gebruiker de naam in het bewaarscherm verandert, moet de nieuwe naam uit het invoerveld gehaald worden. Dat doen we met PROCinvoer.

Er moet nog iets gezegd worden over de padnaam. Die moet twee bewerkingen ondergaan. Eerst moet de naam worden afgesloten met CHR$(0). Dat doen we met FNsluitpad($padnaam%). Maar voor het eigenlijke bewaren kan plaatsvinden, moet het pad compleet worden gemaakt. Vanaf ADFS::HardDisc4.$. tot aan de bestandsnaam Naamloos. Dat doen we met FNzoekpad. Die haalt het hele pad uit blok%!44 en daar kan PROCbewaar mee verder. We hebben er twee functies voor want die hebben we vaker nodig.

De procedure wordt nu:

```
DEF PROCdrop
 PROCinvoer(hsave%,1):REM Haal de bestandsnaam uit het
invoerveld van de savebox
 $padnaam%=A$:REM en zet die via A$ in padnaam
 SYS “Wimp_GetPointerInfo”,,blok%
 !blok%=256
 blok%!16=1:REM de code voor Message_DataSave
 blok%!20=blok%!12:REM eerst de data overnemen
 blok%!24=blok%!16
 blok%!28=!blok%
 blok%!32=blok%!4
 blok%!36=LEN var1$+LEN var2$+LEN var3$+23:REM Bepaal
lengte van de data
 blok%!12=0:REM uw ref. is nog niet bekend
 blok%!40=&XXX:REM het bestandstype
 $(blok%+44)=FNsluitpad($padnaam%)
 SYS “Wimp_SendMessage”,17,blok%,blok%!20,blok%!24
ENDPROC

DEF FNsluitpad(a$):REM Zorgt dat het pad wordt afgesloten
met CHR$0
 WHILE INSTR(a$,”.”)
  a$=MID$(a$,INSTR(a$,”.”)+1)
 ENDWHILE
=a$+CHR$0
```

Als de Filer dat bericht ontvangen heeft, stuurt het een bevestiging, de DataSaveAck. Wij maken daarvoor PROCdatasaveack. Is die bevestiging aangekomen, dan moet het bestand zelf naar de Filer gestuurd worden. Dat doen we met PROCbewaar.

```
WHEN 2:PROCdatasaveack
```

```
DEF PROCdatasaveack
 $padnaam%=FNzoekpad
 PROCbewaar
 blok%!12=blok%!8:REM Verwisseling van de referenties
 blok%!16=3:REM code voor Message_DataLoad
 !blok%=256
 SYS “Wimp_SendMessage”,17,blok%,blok%!20,blok%!24
ENDPROC

DEF FNzoekpad
 a$=””:REM maak leeg voor herhaling van de drag en drop
 letter%=blok%+44:REM Haal het pad uit blok%+44
 WHILE ?letter%<>0:REM Het pad eindigt op CHR$0
  a$=a$+CHR$(?letter%):letter%=letter%+1:REM Zet het pad
in a$ tot CHR$0
 ENDWHILE
 =a$

DEF PROCbewaar
 REM Zet hier de regels die de data vastleggen, bijv:
 x%=OPENOUT($padnaam%)
 BPUT#x%,”Filenaam: “+$padnaam%
 CLOSE#x%
 OSCLI “SetType “+$padnaam%+” XXX”:REM Voeg bestandstype
toe
 !blok%=hsave%:REM Sluit het bewaarvenster
 SYS “Wimp_CloseWindow”,,blok%
ENDPROC
```

Dan nog het Ok-knopje in het bewaarvenster. Roep vanuit PROCmuis op de juiste wijze PROCsnelsave aan. In PROCsnelsave wordt de padnaam gecontroleerd. Staat er een punt in, dan is het pad bekend en wordt PROCsave aangeroepen. Als het pad niet bekend is, wordt er een bericht getoond.

```
DEF PROCsnelsave
 IF INSTR($padnaam%,”.”) THEN PROCbewaar ELSE
PROCmelding(”Sleep het icoon naar de plaats van
bestemming”)
ENDPROC
```

4.5 laden en bewaren met drag and drop: nu het laden

Als je jouw bestand voorzien hebt van een eigen formaat met een eigen sprite, dan zou je daar maar op hoeven te klikken en dan wordt eerst jouw programma geladen en dan het bestand. Hoe gaat dat? We hoeven er vooralsnog geen wijzigingen voor aan te brengen in het programma, maar we moeten wel de !Boot van het programma wijzigen. Dat komt omdat de !Boot het eerste bestand is dat de Filer tegenkomt als het een directory opent. Als het daar al bekend gemaakt wordt met ons bestandstype en daar wordt vermeld wat ermee moet gebeuren, kan zo het programma geladen worden. Het gebeurt met de volgende regel.

```
Set Alias$@RunType_XXX Run <MijnProg$Dir>.!Run
```

Als we willen dat ons bestand niet alleen een hexcode heeft, maar het type ook een naam (voorbeelden staan in het begin van deze Case), dan doen we dat met:

```
Set File$Type_XXX NAAM
```

Als je dit gedaan hebt, kun je op het bestandsicoon klikken. Als de Filer !MijnProg gezien heeft, zal het worden geladen. Hoe ziet de Filer !MijnProg? Door het in een directoryvenster te zetten dat tijdens het starten van het systeem geopend wordt. Of je zet !MijnProg in de bootsequentie bij ‘Look at’.

Dan het laden zelf. Dat gaat nogal direct. De PRM zegt het volgende:
1 de Filer stuurt een DataLoad(3 via 17) naar de applicatie
2 de applicatie laadt het bestand en antwoordt met een DataLoadAck(4 via 18).

Of in onze termen:
De gebruiker sleept het bestandsicoon naar het venster van !MijnProg of naar het icoon op de icoonbalk. Dat wordt door PROContvang, 3 gedetecteerd. We maken daarvoor PROCdataload.

Dan wordt het bestand geladen en wordt “Wimp_SendMessage” met code 4 teruggestuurd.

Het is verstandig om eerst te checken of het bestandstype wel juist is, want als dat niet klopt is verdere actie overbodig.

```
WHEN 3:IF blok%!40=&XXX THEN PROCdataload ELSE
PROCmelding(“Dit is geen bestand voor MijnProg”,1)
```

In het antwoord (met DataLoadAck) moeten eerst de referenties worden verwisseld en dan roepen we “Wimp_SendMessage” aan met berichtcode 4

```
DEF PROCdataload
 !blok%=256
 blok%!12=blok%!8:REM Verwisseling van de referenties
 blok%!16=4:REM code voor Message_DataLoadAck
 SYS “Wimp_SendMessage”,17,blok%,blok%!20,blok%!24
 $padnaam%=FNzoekpad
 PROClaaddata
ENDPROC

DEF PROClaaddata
 REM Zet hier de regels die de data ophalen,bijvoorbeeld
 x%=OPENUP($padnaam%)
  A$=GET$#x%
 CLOSE#x%
 REM verwerk de invoer in A$ verder met PROCicontekst en
 REM PROCinvoer
ENDPROC
```

Tot zover de gewone gang van zaken bij bewaren en laden met drag en drop. Het hele proces is op de volgende pagina in beeld gebracht.

4.6 drag and drop van icoon naar icoon

Het hele protocol is hetzelfde, maar er zijn ook verschillen in procedures of gedeelten daarvan. Dit proces is op de daaropvolgende pagina in beeld gebracht.

Wil je beide combineren, dus laden/bewaren en van icoon naar icoon, zul je oplossingen moeten zoeken voor de verschillen door programmaregels te combineren en voorwaarden te bepalen.

Drag&drop laden en bewaren:

In PROCbegin moet het volgende worden toegevoegd:

```
DIM list% 23, padnaam% 256
!list%=1:list%!4=2:list%!8=3:list%!12=&502:list%!16=&503:list%!20=0
$padnaam%="Naamloos"
```

De drag start met de muis (select ingedrukt houden en slepen)
in PROCmuis met klik%=64

```
IF venster%=hsave% AND icoon%=0 AND klik%=64 PROCdrag(hsave%,0)
IF venster%=hsave% AND icoon%=3 PROCsnelsave
```

Om de dragbox te zien, wordt PROCdrag aangeroepen

```
DEF PROCdrag(venster%,icoon%)
 $padnaam%=""
 !blok%=venster%
 SYS "Wimp_GetWindowState",,blok%
 x%=blok%!4-blok%!20
 y%=blok%!16-blok%!24
 blok%!4=icoon%
 SYS "Wimp_GetIconState",,blok%
 blok%!4=5
 blok%!8=x%+blok%!8
 blok%!12=y%+blok%!12
 blok%!16=x%+blok%!16
 blok%!20=y%+blok%!20
 blok%!24=0
 blok%!28=0
 blok%!32=&7FFFFFFF
 blok%!36=&7FFFFFFF
 SYS "Wimp_DragBox",,blok%
ENDPROC
```

```
DEF PROCsnelsave
 IF INSTR($padnaam%,".") THEN PROCbewaar ELSE
  PROCmelding("Sleep het icoon naar de plaats
  van bestemming",1)
ENDPROC
```

```
DEF PROCdrop
 PROCinvoer(hsave%,1)
 $padnaam%=A$
 SYS "Wimp_GetPointerInfo",,blok%
 !blok%=256
 blok%!16=1
 blok%!20=blok%!12
 blok%!24=blok%!16
 blok%!28=!blok%
 blok%!32=blok%!4
 blok%!36=4000:REM geschatte bestandsgrootte
 blok%!12=0
 blok%!40=&FFD:REM bestandstype
 $(blok%+44)=FNsluitpad($padnaam%)
 SYS "Wimp_SendMessage",17,blok%,blok%!20,blok%!24
ENDPROC
```

In de Wimp-poll wordt de drag bij event 7 gesignaleerd

```
WHEN 0:
WHEN 1:
WHEN 2:PROCopenvenster
WHEN 3:PROCsluitvenster
WHEN 4:
WHEN 5:
WHEN 6:PROCmuis(blok%!0,blok%!4,blok%!8,blok%!12,blok%!16)
WHEN 7:PROCdrop
WHEN 8:PROCtoets(blok%!0,blok%!4,blok%!24)
WHEN 9:PROCmenukeuze(menu%)
WHEN 17,18:PROContvang
WHEN 19:PROCmelding("Bewaren mislukt",1)
```

```
DEF FNsluitpad(a$)
 WHILE INSTR(a$,".")
 a$=MID$(a$,INSTR(a$,".")+1)
 ENDWHILE
=a$+CHR$0
```

```
DEF FNzoekpad
 a$=""
 letter%=blok%+44
 WHILE ?letter%<>0
  a$=a$+CHR$(?letter%):letter%=letter%+1
 ENDWHILE
=a$
```

Nu wordt het berichtenverkeer gestart via event 17,18

```
DEF PROContvang
 CASE blok%!16 OF
  WHEN 0:klaar%=TRUE
  WHEN 2:PROCdatasaveack
  WHEN 3:PROCdataload
 ENDCASE
ENDPROC
```

```
DEF PROCdataload
 !blok%=256
 blok%!12=blok%!8
 blok%!16=4
 SYS "Wimp_SendMessage",17,blok%,blok%!20,blok?!24
 $padnaam%=FNzoekpad
 PROClaaddata
ENDPROC
```

```
DEF PROCdatasaveack
 $padnaam%=FNzoekpad
 PROCbewaar
 blok%!12=blok%!8
 blok%!16=3:!blok%=256
 SYS "Wimp_SendMessage",11,blok%,blok%!20,blok%!24
ENDPROC
```

```
DEF PROClaaddata(padnaam$)
 f%=OPENUP(padnaam$)
   REM hier komen de variabelen
   REM die gevuld moeten worden
   REM met waarden uit het bestand
 CLOSE#f%
ENDPROC
```

```
DEF PROCbewaar
 f%=OPENOUT($padnaam%)
   REM geef hier de regels
   REM om de variabelen in
   REM het bestand te schrijven
 CLOSE#f%
 !blok%=hsave%
 SYS "Wimp_CloseWindow",,blok%
ENDPROC
```

Drag&drop van icoon naar icoon:

In PROCbegin moet het volgende worden toegevoegd:

```
DIM list% 23, padnaam% 256
!list%=1:list%!4=2:list%!8=3:list%!12=&502:list%!16=&503:list%!20=0
$padnaam%="Wimpscrap"
```

De drag start met de muis (select ingedrukt houden en slepen)
in PROCmuis met klik%=64

```
IF venster%=hmain% AND icoon%=6 AND klik%=64 PROCdrag(hmain%,6)
```

Om de dragbox te zien, wordt PROCdrag aangeroepen

```
DEF PROCdrag(venster%,icoon%)
 $padnaam%=""
 !blok%=venster%
 SYS "Wimp_GetWindowState",,blok%
 x%=blok%!4-blok%!20
 y%=blok%!16-blok%!24
 blok%!4=icoon%
 SYS "Wimp_GetIconState",,blok%
 blok%!4=5
 blok%!8=x%+blok%!8
 blok%!12=y%+blok%!12
 blok%!16=x%+blok%!16
 blok%!20=y%+blok%!20
 blok%!24=0
 blok%!28=0
 blok%!32=&7FFFFFFF
 blok%!36=&7FFFFFFF
 SYS "Wimp_DragBox",,blok%
ENDPROC
```

In de Wimp-poll wordt de drag bij event 7 gesignaleerd

```
WHEN 0:
WHEN 1:
WHEN 2:PROCopenvenster
WHEN 3:PROCsluitvenster
WHEN 4:
WHEN 5:
WHEN 6:PROCmuis(blok%!0,blok%!4,blok%!8,blok%!12,blok%!16)
WHEN 7:PROCdrop
WHEN 8:PROCtoets(blok%!0,blok%!4,blok%!24)
WHEN 9:PROCmenukeuze(menu%)
WHEN 17,18:PROContvang
WHEN 19:PROCmelding("Bewaren mislukt",1)
```

Nu wordt het berichtenverkeer gestart via event 17,18

```
DEF PROContvang
 CASE blok%!16 OF
  WHEN 0:klaar%=TRUE
  WHEN 1:PROCdatasave
  WHEN 2:PROCdatasaveack
  WHEN 3:PROCdataload
 ENDCASE
ENDPROC
```

```
DEF PROCdataload
 !blok%=256
 blok%!12=blok%!8
 blok%!16=4
 SYS "Wimp_SendMessage",17,blok%,blok%!20,blok?!24
 $padnaam%=FNzoekpad
 PROClaaddata
ENDPROC
```

```
DEF PROCdatasaveack
 $padnaam%=FNzoekpad
 PROCbewaar
 blok%!12=blok%!8
 blok%!16=3:!blok%=256
 SYS "Wimp_SendMessage",11,blok%,blok%!20,blok%!24
ENDPROC
```

nieuw

```
DEF PROCdatasave
 !blok%=256
 blok%!12=blok%!8
 blok%!16=2
 $padnaam%="<Wimp$Scrap>"
 blok%!36=-1
 blok%!40=&FFF
 $(blok%+44)=FNsluitpad($padnaam%)
 SYS "Wimp_SendMessage",17,blok%,blok%!20,blok%!24
ENDPROC
```

```
DEF PROCdrop
 PROCinvoer(hmain%,6)
 REM de inhoud van A$ wordt verstuurd
 SYS "Wimp_GetPointerInfo",,blok%
 !blok%=256
 blok%!16=1
 blok%!20=blok%!12
 blok%!24=blok%!16
 blok%!28=!blok%
 blok%!32=blok%!4
 blok%!36=4000:REM geschatte bestandsgrootte
 blok%!12=0
 blok%!40=&FFF:REM bestandstype
 $(blok%+44)=FNsluitpad($padnaam%)
 SYS "Wimp_SendMessage",17,blok%,blok%!20,blok%!24
ENDPROC
```

```
DEF FNsluitpad(a$)
 WHILE INSTR(a$,".")
 a$=MID$(a$,INSTR(a$,".")+1)
 ENDWHILE
=a$+CHR$0
```

```
DEF FNzoekpad
 a$=""
 letter%=blok%+44
 WHILE ?letter%<>0
  a$=a$+CHR$(?letter%):letter%=letter%+1
 ENDWHILE
=a$
```

```
DEF PROClaaddata(padnaam$)
 f%=OPENUP(padnaam$)
   A$=GET$#f%
 CLOSE#f%
OSCLI "Delete "+$padnaam%
REM zet A$ in het juiste icoon  en
REM verwerk de inhoud van A$
ENDPROC
```

anders

```
DEF PROCbewaar
 f%=OPENOUT($padnaam%)
   BPUT#f%,A$
   REM de A$ die in PROCdrop
   REM gevuld is met inhoud icoon
 CLOSE#f%
 OSCLI "SetType "+$padnaam%+" FFF"
ENDPROC
```

Afbeeldingen in een icoon Case 5

Een eigenaar van een fitnesscentrum wil zijn administratie met een A9 gaan doen. Die is overdag in het centrum maar hij neemt hem ‘s avonds mee naar huis. Kan hij fijn verder werken en hij hoeft niet bang te zijn dat zijn computer gestolen wordt.

Hij wil een programma voor het bijhouden van klantgegevens, bezoeken en betalingen. In dat programma wordt een database opgebouwd met, ondermeer, naam, voorletters, voornamen, roepnaam, straat, huisnummer, postcode, woonplaats, telefoon vast, telefoon mobiel, en e-mailadres. Verder moet er een venster zijn waarin betalingsgegevens worden bijgehouden. Daarin wordt het gestorte bedrag ingevoerd en per training wordt daar een bedrag afgetrokken. Het resterende bedrag wordt getoond en is dat onder een bepaald minimum, dan vindt er een waarschuwing plaats in het hoofdvenster.

De eigenaar wil nu in het hoofdvenster een plaats waar een fotootje wordt afgebeeld. Zo kan hij zijn klanten beter herkennen. Op de balie staat een cameraatje op een driepootje en via een usb-kabeltje is het verbonden met het A9’tje. Als er een foto wordt gemaakt, verschijnt die in het hoofdvenster in het juiste icoon. Per klant hoeft de foto maar eenmalig gemaakt te worden.

De vraag is dus: hoe zetten we een foto in het venster. Of algemener: hoe zetten we een afbeelding in een icoon.

Denkstap
Benodigdheden
Uitwerking