# Paradox voor Windows

Versie 4.5

---

# ObjectPAL™-handboek

# INHOUD

# VOORBEELDEN

# TABELLEN

# AFBEELDINGEN

HOOFDSTUK 1

# Inleiding

ObjectPAL™ is de geïntegreerde programmeertaal van Paradox voor Windows. U kunt ObjectPAL gebruiken om complete Windows-applicaties te ontwikkelen of om aan een Paradox-applicatie mogelijkheden toe te voegen die interactief niet beschikbaar zijn.

Als u een beginnend programmeur bent, zult u ontdekken dat u met ObjectPAL gemakkelijk interactieve applicaties kunt verbeteren. Enkele voorbeelden:

- U kunt knoppen toevoegen waarmee regelmatig terugkerende handelingen kunnen worden uitgevoerd.
- Als uw database veel gegevens bevat, kunt u een dialoogvenster maken, zodat de gebruiker bij gecompliceerde zoekopdrachten gemakkelijker de gewenste records vindt.
- Uw applicatie kan een betrouwbare module voor gegevensinvoer vereisen. In dat geval kunt u een bewerkingsroutine maken die de gegevensinvoer van de gebruiker controleert en bij foutieve invoer reageert.
- Ter verfraaiing van uw applicatie kunt u animatie-effecten maken met ObjectPAL.

Als u nog niet eerder hebt geprogrammeerd en niet zeker weet of u ObjectPAL wilt leren, kunt u de zelfstudie (Hoofdstuk 2 tot en met 7) bekijken om een idee te krijgen van wat u met ObjectPAL kunt doen. U krijgt de smaak waarschijnlijk snel te pakken.

## Voordat u met ObjectPAL gaat werken

ObjectPAL en Paradox zijn sterk geïntegreerd. Als u veel van Paradox weet, kunt u deze kennis toepassen in uw ObjectPAL-programma's.

**Belangrijk** Als u zoveel mogelijk kennis wilt opdoen uit deze handleiding, kunt u beter eerst het *Handboek* lezen en interactief ervaring opdoen met Paradox, zodat u weet hoe u:

- Tabellen, formulieren en rapporten maakt
- De TurboBalk gebruikt om ontwerpobjecten te plaatsen
- Interactief werkt met tabelframes en multi-record objecten
- Objecten benoemt
- Objecten inspecteert en objectkenmerken instelt
- De werkdirectory instelt en wijzigt
- Queries maakt met Query By Example (QBE)
- Aliassen toewijst
- Tabellen sorteert

Als u vertrouwd bent met deze handelingen en begrippen, is ObjectPAL gemakkelijker te leren.

Als u Paradox nog niet hebt geïnstalleerd en geconfigureerd, raadpleeg dan *Aan de slag* voor aanwijzingen.

*U leert ObjectPAL het best kennen door het te gebruiken.*

Deze handleiding behandelt niet alle aspecten van het programmeren, maar introduceert programmeerconcepten die van toepassing zijn op ObjectPAL. U kunt het best de voorbeelden doorwerken en zelf experimenteren. U leert ObjectPAL het best kennen door het te gebruiken. Vaak kunt u een taak met ObjectPAL op verschillende manieren uitvoeren. Door te experimenteren ontdekt u vanzelf welke manier u het meest ligt.

### Opmerking voor PAL-programmeurs

Als u een ervaren PAL-programmeur bent, zult u ontdekken dat ObjectPAL in veel opzichten afwijkt. Bepaalde aspecten van de databaseprogrammering zijn echter hetzelfde, zoals het werken met gegevens in tabellen, records en velden, het gebruik van Query By Example en de controle van de toegang tot gegevens. In Appendix A vindt u een opsomming van de belangrijkste verschillen tussen PAL en ObjectPAL.

## Gebruik van de handleiding

Dit boek bestaat uit een zelfstudie, gevolgd door gedetailleerde informatie en voorbeelden voor meer ervaren gebruikers.

De zelfstudie is bedoeld om gebruikers die geen of weinig programmeerervaring hebben op eenvoudige wijze op weg te helpen

met ObjectPAL. Voor de ervaren programmeur is deze zelfstudie een snelle inleiding op de ObjectPAL-taal. Aan de hand van de voorbeelden leert u Paradox voor Windows-applicaties maken met knoppen, dialoogvensters, applicaties, validiteitscontroles en controles op sleutelinbreuk. De voorbeelden kunnen het beste worden doorlopen in de volgorde waarin zij worden gepresenteerd. U kunt de zelfstudie achteraf altijd raadplegen voor specifieke code.

De rest van dit boek bespreekt informatie en concepten die u nodig hebt voor het ontwerpen van Paradox-applicaties. In deze handleiding wordt ervan uitgegaan dat u vertrouwd bent met de interactieve kant van Paradox. Met de "interactieve kant van Paradox" wordt alles bedoeld wat u in Paradox kunt bereiken zonder ObjectPAL.

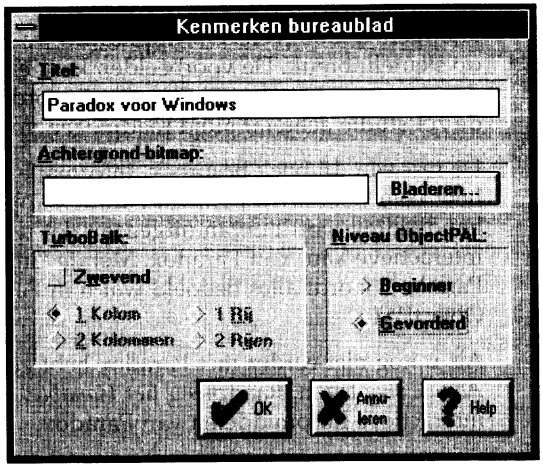
Als u nog niet eerder met Paradox hebt gewerkt, kunt u het best eerst *Aan de slag* lezen. Raadpleeg het *Handboek* voor informatie over de interactieve hulpmiddelen van Paradox.

## Het ObjectPAL-niveau instellen

Door een bepaald bureaubladkenmerk in te stellen, kunt u de ObjectPAL Integrated Development Environment (IDE) zo instellen dat u alles in de taal te zien krijgt of alleen de belangrijkste elementen (de standaardinstelling). Als u het ObjectPAL-niveau wilt instellen, kiest u 'Kenmerken | Bureaublad' om het dialoogvenster 'Kenmerken bureaublad' te openen (zie Afbeelding 1-1). Kies vervolgens 'Beginner' in het paneel 'Niveau ObjectPAL' als u alleen de belangrijkste elementen wilt zien of 'Gevorderd' als u alles wilt zien. Deze instelling beïnvloedt alleen de IDE, *de code blijft ongewijzigd*. De code wordt bij de twee instellingen hetzelfde uitgevoerd. Als het ObjectPAL-niveau is ingesteld op 'Beginner', kunt u toch gebruik maken van gevorderde elementen.

***Belangrijk*** Als u voor het eerst met ObjectPAL werkt, kunt u het beste de instelling 'Beginner' gebruiken, de standaardinstelling voor het ObjectPAL-niveau. U hebt dan meer overzicht zodat u gemakkelijker de elementen kunt kiezen die u nodig hebt. In de zelfstudie wordt ervan uitgegaan dat u het ObjectPAL-niveau hebt ingesteld op 'Beginner'. De rest van het boek gaat uit van de instelling 'Gevorderd'.

Afbeelding 1-1 Instellen van het ObjectPAL-niveau




In de zelfstudie wordt ervan uitgegaan dat het ObjectPAL-niveau is ingesteld op 'Beginner'; in de rest van deze handleiding wordt uitgegaan van de instelling 'Gevorderd'

## Lettertypes

In Tabel 1-1 wordt een overzicht gegeven van de lettertypes die in dit boek worden gebruikt.

Tabel 1-1 Lettertypes

| Conventie | Toepassing | Voorbeelden |
|---|---|---|
| **Vet** | Methodenamen en berichten die Paradox weergeeft | **insertRecord,** Paradox toont het bericht **Ongeldige invoer in sleutelveld** |
| *Cursief* | Namen van Paradox-objecten, termen uit de woordenlijst, variabelen, benadrukte woorden | de tabel *Antwrd, zoekKnop, zoekWaarde* |
| KAPITALEN | DOS-bestanden en -directories, gereserveerde woorden, operatoren, soorten queries | directory VOORBD, KLANT.DB, C:\WINDOWS |
| Begin-kapitalen | Velden, menu-opdrachten, objectnamen; tussen enkele aanhalingstekens | 'Prijs'-veld, opdracht 'Formulier \| Gegevens tonen' |
| *Toetsopdruk* | Toetsen op uw toetsenbord | *F1, Enter* |
| `Typemachine-letter` | ObjectPAL-code | `mijnTabl.open("plaats.db")` |
| **Invoerletter** | Tekst die u invoert | **jan - jun, 20-07-92** |
|  | Informatie voor beginnende gebruikers | |
| ***Belangrijk*** | Belangrijke informatie | |

In tekst (in tegenstelling tot codevoorbeelden) verwijst deze handleiding alleen naar de namen van methodes en procedures en dus niet naar de volledige syntaxis. Stel dat in de tekst de methode **attach** wordt genoemd, die is gedefinieerd voor het type Table. Dat is dan een verwijzing naar de methode waarvan de volledige syntaxis is:

**attach** ( const ***tableName*** String ) Logical

Voor de volledige syntaxis van ObjectPAL-methodes en -procedures kunt u de online ObjectPAL Help of de *Naslaggids* raadplegen.

Onderstaande tabel bevat een beknopt overzicht van de notatie van ObjectPAL-syntaxis :

Tabel 1-2 Lettertypes voor syntaxis

| Lettertype | Element | Voorbeeld | Betekenis |
|---|---|---|---|
| Normaal font | Sleutelwoord | setPosition | Exact overnemen. |
| *Cursief* | Invullen | *tabelVar* | Vervangen door een uitdrukking. |
| { \| } (accolades en streepje) | Keuze | { Yes \| No } | U *moet* een van de elementen kiezen die door een verticale streep worden gescheiden. |
| [ ] (vierkante haakjes) | Optioneel | [ , *tabelVar2* ] [ ELSE ] | U *kunt* kiezen of u dit opneemt. |
| * (asterisk) | Herhaling | [ , *tabelVar2* ] * | U kunt dit argument herhalen. |

# Het gebruik van de online ObjectPAL Help

De online ObjectPAL Help biedt uitgebreide informatie over ObjectPAL, waaronder

- De ideeën die aan ObjectPAL ten grondslag liggen.
- Een complete referentie voor de taal (de elementen van de taal, de methodes en procedures in de ObjectPAL run-time bibliotheek en de ObjectPAL constanten).
- Voorbeelden van ObjectPAL-code die u in uw eigen applicaties kunt plakken.

U kunt de ObjectPAL Help te allen tijde oproepen op de volgende wijze:

1. Druk op *F1* om de Helpapplicatie te openen.
2. Kies 'Bestand | Openen' om het dialoogvenster 'Openen' te openen.

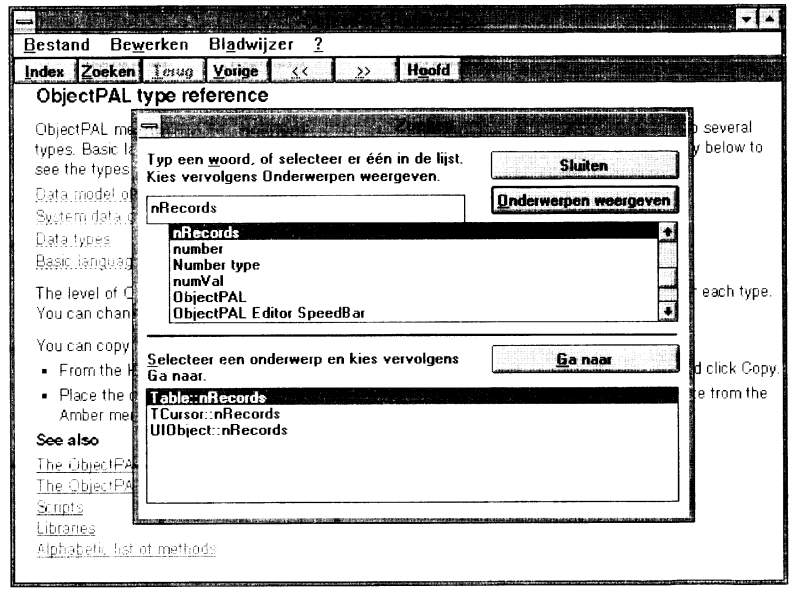
3. Kies PAL.HLP en klik daarna op 'OK' om het ObjectPAL Helpbestand te openen.

Als u weet waarnaar u zoekt, kunt u werken met de zoekfunctie, zoals wordt getoond in Afbeelding 1-2. Als u niet precies weet waarnaar u op zoek bent, kunt u de helpfuncties gebruiken (zoals klikken op een onderstreept woord) of door het bestand bladeren. U kunt code uit de voorbeelden in Help naar het Klembord kopiëren en deze in uw eigen applicaties plakken. U hoeft de code dan niet opnieuw te typen en u verkleint daarmee de kans op typfouten.

Afbeelding 1-2 Het ObjectPAL Helpsysteem gebruiken

Gebruik de zoekfunctie om een bepaald onderwerp op te zoeken.

Gebruik de andere helpfuncties (zoals klikken op een onderstreept woord) om door het ObjectPAL Helpbestand te bladeren.




# Het gebruik van de voorbeeldbestanden

De voorbeeldbestanden bevatten één volledige applicatie (Duikplan, ook MAST genoemd) en verschillende mini-applicaties. Alle voorbeeldapplicaties bevatten interactieve online helpfuncties, waarin wordt uitgelegd hoe u de voorbeelden moet gebruiken, en help-systemen met uitleg over de ObjectPAL-code die aan de objecten is gekoppeld. Als u met deze applicaties werkt, kunt u hulp opvragen door op *F1* te drukken of een optie te kiezen uit het helpmenu (?) op de menubalk. In Duikplan kunt u ook helpinformatie over ObjectPAL opvragen door een object te inspecteren (rechts klikken) en 'Code Help' te kiezen in het pop-up menu. Als u 'Code Help' kiest, wordt er een Helpvenster geopend met een overzicht van de methodes en

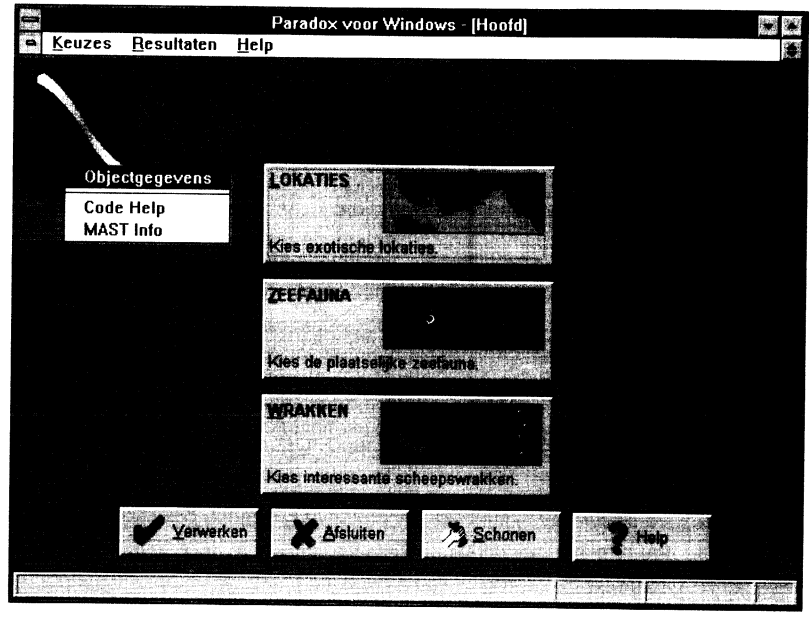
procedures die zijn gekoppeld aan het geïnspecteerde object (zie Afbeelding 1-3). In het Helpvenster kunt u vervolgens het onderdeel kiezen waarin u geïnteresseerd bent en zowel de code als de verklarende tekst laten weergeven.

***Opmerking*** De applicatie Duikplan is een hulpmiddel bij het leren van ObjectPAL. De applicatie demonstreert veel technieken en werkwijzen voor het gebruik van ObjectPAL. Deze applicatie is echter niet bedoeld als voorbeeld van een realistisch programma.

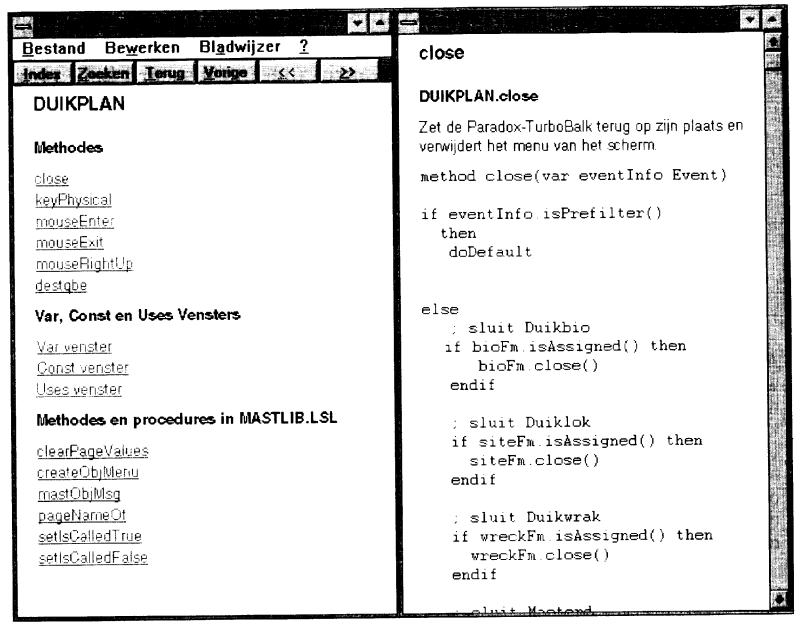
Afbeelding 1-3 Het Helpsysteem gebruiken in de voorbeeldapplicatie

Als u met een voorbeeldapplicatie werkt, kunt u te allen tijde met de rechter muisknop op een object klikken om een pop-up menu te openen. Kies 'Code Help' om een helpvenster te openen dat een overzicht geeft van de ObjectPAL-code die is gekoppeld aan het object.

(In Duikplan vertegenwoordigt de duikvlag het formulier)



Als u rechts op een object klikt en 'Code Help' kiest, wordt een Helpvenster geopend met een overzicht van alle ObjectPAL-code die aan dat object is gekoppeld. Kies een methode uit dit overzicht om een tweede venster te openen met de broncode van de methode en met verklarende tekst.



DEEL I

# ObjectPAL zelfstudie

Dit deel van deze handleiding bevat een zelfstudie die zo is opgezet dat u zo snel mogelijk aan de slag kunt met ObjectPAL. Aan de hand van stapsgewijze instructies wordt getoond hoe u de meest voorkomende database programmeertaken met ObjectPAL kunt uitvoeren.

Deel I bevat de volgende hoofdstukken:

- In Hoofdstuk 2, "Knoppen programmeren", wordt getoond hoe u ObjectPAL-code aan een knop koppelt, zodat de code wordt uitgevoerd als op de knop wordt geklikt.
- In Hoofdstuk 3, "Handelingen", worden de grondslagen van het programmeren in ObjectPAL besproken.
- In Hoofdstuk 4, "Invoer en uitvoer", wordt getoond hoe u informatie kunt opvragen bij de gebruiker, hoe u informatie aan de gebruiker toont en hoe u deze informatie verwerkt.
- In Hoofdstuk 5, "Gegevensinvoer valideren", worden technieken behandeld die verifiëren of de gegevens die de gebruiker invoert geldig zijn.
- In Hoofdstuk 6, "Andere formulieren besturen", wordt besproken hoe ObjectPAL kan worden gebruikt om een formulier vanuit een ander formulier te besturen en hoe een formulier als dialoogvenster kan worden gebruikt.
- Hoofdstuk 7, "Buiten het gegevensmodel werken", is bedoeld voor programmeurs die een van de meer geavanceerde mogelijkheden van ObjectPAL willen uitproberen: het werken met tabellen die geen onderdeel uitmaken van het gegevensmodel van dat formulier.

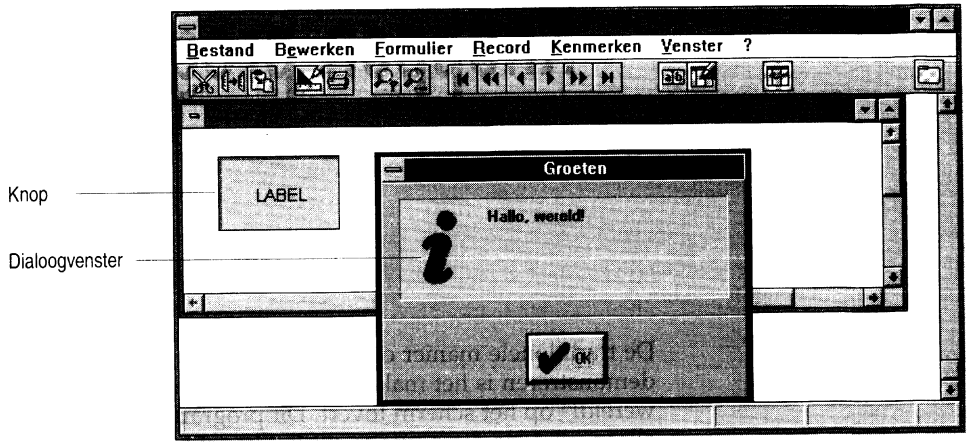
HOOFDSTUK 2

# Knoppen programmeren

De traditionele manier om de werking van een nieuwe taal te demonstreren is het maken van een programma dat de tekst "Hallo, wereld!" op het scherm tovert. Dit programma bestaat gewoonlijk alleen uit code die nodig is om het bericht "Hallo, wereld!" op het scherm weer te geven. De klassieke uitvoering van het programma "Hallo, wereld!" is niet interactief. U start het programma en het bericht verschijnt. Dat is alles.

ObjectPAL daarentegen is een taal waarmee u interactieve, actiegestuurde applicaties kunt maken. In de volgende versie van de applicatie "Hallo, wereld!" bepaalt de gebruiker wanneer het bericht wordt weergegeven en wanneer het verdwijnt. U programmeert een knop die een dialoogvenster weergeeft, maar de code wordt alleen uitgevoerd als de gebruiker op de knop klikt en het dialoogvenster blijft geopend totdat de gebruiker het sluit. In Voorbeeld 2-1 ziet u hoe u dit doet. In Afbeelding 2-1 ziet u de applicatie in actie.

Afbeelding 2-1 Hallo, wereld!




Als u de instructie **msgInfo("Groeten", "Hallo, wereld!")** toevoegt aan de ingebouwde **pushButton**-methode van de knop, verschijnt het dialoogvenster zodra u op de knop klikt. Het dialoogvenster blijft open totdat u het sluit.

***Opmerking*** Verander uw werkdirectory in PDOXWIN\VOORBD als u de voorbeelden in dit handboek wilt volgen.

## Voorbeeld 2-1 De applicatie Hallo, wereld!

1. Maak eerst een nieuw formulier. Kies 'BestandINieuwIFormulier' op het bureaublad om het formulierontwerp te activeren. U ziet verschillende dialoogvensters na elkaar verschijnen. Accepteer de standaardinstellingen in elk dialoogvenster om een leeg formulier te maken. Bij de ontwikkeling van Paradox-applicaties bestaat het meeste werk uit het plaatsen van objecten in formulieren en het schrijven van de ObjectPAL-code die aangeeft hoe de objecten op acties reageren.


2. Klik op het knophulpmiddel om een knop te maken. Plaats de knop ergens op het formulier. U hoeft zich nu nog niet bezig te houden met het label van de knop.

3. Start het formulier door op de knop 'Gegevens tonen' te klikken of door 'FormulierIGegevens tonen' te kiezen.

4. Klik op de knop die u hebt gemaakt, en kijk wat er gebeurt. Het uiterlijk van de knop verandert. Het lijkt of de knop wordt ingedrukt en weer wordt losgelaten. U hebt helemaal geen code geschreven, dus hoe is dit mogelijk? Knoppen zakken standaard naar beneden en springen weer terug als erop wordt geklikt. In de volgende voorbeelden schrijft u ObjectPAL-code om deze knop meer te laten doen.

5. Klik op de knop 'Ontwerpen' of kies 'FormulierlOntwerpen' om door te gaan met het ontwerp van uw formulier.

# Ingebouwde methodes

Elk object op een formulier (ook het formulier zelf) bevat ingebouwde methodes die worden uitgevoerd in antwoord op acties. Paradox interpreteert de actie en roept de juiste ingebouwde methode aan die is gekoppeld aan het doelobject. Toen u op de knop klikte, reageerde deze. Hiervoor hoefde u niets te programmeren. In alle Paradox-objecten zijn namelijk standaardmethodes ingebouwd.

# Standaardgedrag veranderen

Als u de manier wilt veranderen waarop een knop reageert als u erop klikt, koppelt u uw eigen code aan de ingebouwde **pushButton**-methode van de knop. In Voorbeeld 2-2 ziet u hoe u de ObjectPAL-Editor opent.

### Voorbeeld 2-2 Een venster van de ObjectPAL-Editor openen

1. Inspecteer de knop om de kenmerken te bekijken. (Als u een object wilt inspecteren, klikt u er met de rechtermuisknop op, selecteert u het object en drukt u op ***F6*** of selecteert u het object en kiest u 'KenmerkenlHuidig object'.

*Ingebouwde methodes definiëren hoe objecten op acties reageren*

2. Kies 'Methodes' om het methodevenster te openen dat een overzicht geeft van de ingebouwde methodes van de knop.

***Snelle manier***

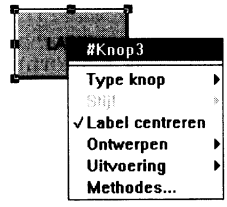
Selecteer de knop en druk op ***Ctrl-Spatiebalk*** om het methodevenster te openen.

3. U wilt definiëren wat er gebeurt als er op de knop wordt geklikt. Selecteer dus pushButton uit de lijst met methodes en kies vervolgens 'OK'. Er wordt een venster van de ObjectPAL-Editor geopend dat de volgende code bevat:

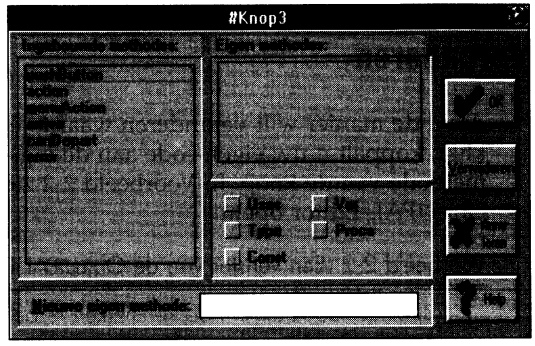
```
method pushButton (var eventInfo Event)

endmethod
```

In Afbeelding 2-2 worden al deze stappen getoond.

Afbeelding 2-2 Inspecteer het object, kies een methode, open een Editor-venster




Inspecteer eerst de knop en kies vervolgens 'Methodes' in het menu om het methodevenster te openen.




Kies vervolgens pushButton om de **pushButton**-methode te bewerken en kies 'OK' om een venster van de ObjectPAL-Editor te openen.


Gebruik vervolgens de ObjectPAL-Editor om de methode te bewerken.

Deze methode doet nog niet veel. (In de paragraaf "Werking", verderop in dit hoofdstuk, wordt de betekenis van *var eventInfo Event* uitgelegd. U kunt deze code voorlopig negeren.) Het is belangrijk te begrijpen dat Paradox op een standaardmanier reageert op alle objecten die u maakt, en op alle acties die van invloed zijn op een object. Soms is de standaardreactie van Paradox zichtbaar, zoals een knop die beweegt als u erop klikt. In andere gevallen is de standaardreactie niet zichtbaar, maar toch noodzakelijk.

Bovendien is het van belang te beseffen dat de standaardreactie alleen optreedt als het object een actie ontvangt. De knop beweegt bijvoorbeeld pas als u erop klikt.

# Uw eigen code koppelen

De volgende stap is uw eigen code aan deze methode te koppelen. Als u code aan een methode koppelt, wordt de standaardreactie *niet* uitgeschakeld. Eerst wordt uw code uitgevoerd en vervolgens de standaardcode.

### Voorbeeld 2-3 Uw eigen code koppelen

1. Als er nog geen venster van de ObjectPAL-Editor open is, open dan een venster om de **pushButton**-methode van de knop te bewerken.
2. Voeg code toe aan de **pushButton**-methode, zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   msgInfo("Groeten", "Hallo, wereld!")
endmethod
```

3. Klik op de knop 'Syntaxis controleren' (of kies 'Taal|Syntaxis controleren') om uw code te controleren op syntaxisfouten en spelfouten. Als er fouten voorkomen, wordt dit aangegegeven door een bericht op de statusbalk. Als de code geen fouten bevat, verschijnt het bericht **Geen syntaxisfouten** op de statusbalk en worden uw veranderingen in het geheugen opgeslagen.

***Snelle manier***

Klik met de rechtermuisknop ergens in het Editor-venster om het menu 'Taal' te openen.

4. Klik op de knop 'Gegevens tonen' om het formulier te starten.
5. Klik op de knop die u hebt gemaakt, om uw groet in een dialoogvenster weer te geven.
6. Kies 'OK' om het dialoogvenster te sluiten.

7. Ga terug naar het formulierontwerpvenster en kies 'Bestand|Opslaan' om de veranderingen op schijf op te slaan. Geef dit formulier de naam HALLO.FSL.

*Objecten van het ene naar het andere formulier kopiëren via het Klembord*

U hebt nu een applicatie die een bericht weergeeft in een dialoogvenster als u op een knop klikt (en alleen dan). De applicatie wacht vervolgens totdat u het dialoogvenster sluit. Bovendien kunt u deze knop naar het Klembord kopiëren en deze in een ander Paradox-formulier plakken. De knop werkt dan op exact dezelfde manier.

## Werking

De code die u hebt geschreven, werkt als volgt:

```
method pushButton(var eventInfo Event)
```

De eerste regel geeft aan dat de methode **pushButton** heet, dat **pushButton** één argument heeft, namelijk *eventInfo*, van het type *Event*, en dat *eventInfo* een variabele is die als verwijzing wordt

doorgegeven (aangegeven door het sleutelwoord *var*). Deze instructie (die Paradox standaard produceert) bevat belangrijke informatie voor ObjectPAL. U hoeft hier nu geen aandacht aan te besteden.

De volgende regel roept de procedure **msgInfo** aan.

```
msgInfo("Groeten", "Hallo, wereld!")
```

**msgInfo** is een procedure uit de run-time bibliotheek van ObjectPAL, een verzameling van voorgedefinieerde methodes en procedures die werken op objecten of gegevens van een bepaald type. **msgInfo** heeft twee *argumenten* (soms *parameters* genoemd). Het eerste argument, "Groeten", bevat de tekst die op de titelbalk van het dialoogvenster moet worden weergegeven, en het tweede argument, "Hallo, wereld!", bevat de tekst die in het dialoogvenster zelf moet verschijnen. Dit dialoogvenster is *modaal*. Dat wil zeggen dat het dialoogvenster geopend blijft tot de gebruiker het sluit en dat de gebruiker het moet sluiten voordat hij door kan werken met het formulier.

De laatste regel markeert het einde van de methode.

```
endmethod
```

De ingebouwde code wordt standaard vlak vóór deze regel uitgevoerd.

## Samenvatting

In deze les hebt u een eerste indruk gekregen van wat er komt kijken bij het maken van een Paradox-applicatie. Dit zijn de basisstappen:

1. Plaats objecten op een formulier. Bij elk object hoort ingebouwde code, zodat formulieren ook kunnen worden gestart als u zelf geen code schrijft.
2. Start het formulier en kijk hoe de objecten zich standaard gedragen.
3. Bepaal welke objecten iets anders of iets meer moeten doen.
4. Bepaal wat elk object moet doen en wanneer.
5. Koppel uw eigen code aan de juiste ingebouwde methode(s).
6. Start het formulier en kijk of het doet wat u had verwacht. Spoor indien nodig fouten op.

HOOFDSTUK 3

# Handelingen

De volgende lessen zijn gebaseerd op één applicatie: een formulier voor het invoeren van klantgegevens. De lessen in dit hoofdstuk laten zien

- Hoe code die is gekoppeld aan één object van invloed kan zijn op een ander object
- Hoe u met de UIObject-methode **action** handelingen initieert
- Hoe u met de ingebouwde **action**-methode op een handeling reageert
- Hoe u met de UIObject-methode **locate** waarden zoekt in een tabel

## Fasen in het schrijven van ObjectPAL-toepassingen

In het algemeen worden ObjectPAL-applicaties in de volgende fasen gemaakt:

1. Verzamel uw gegevens.

   Maak de tabellen en vul deze met gegevens. Als u een multi-tabel applicatie maakt, bepaalt u uw gegevensmodel voordat u de tabellen maakt.

2. Maak het formulier.

   Hoewel u scripts kunt schrijven en code kunt opslaan in bibliotheken, wordt het grootste gedeelte van de ObjectPAL-code aan objecten op formulieren gekoppeld. De beste manier om te beginnen is het formulier te maken, de objecten te plaatsen en het formulier te starten. Kijk hoe Paradox zich standaard gedraagt als er geen ObjectPAL-code is gekoppeld. U zult ontdekken dat de ingebouwde code in de meeste gevallen doet wat u wilt. Terwijl u het standaardgedrag bekijkt, kunt u bepalen welke objecten iets

anders of iets meer moeten doen. Daarna kunt u doorgaan met de volgende fase.

3. Koppel ObjectPAL-code aan de ingebouwde methodes van het object.

   Wijzig het gedrag van een object door code aan de juiste ingebouwde methode te koppelen. Aangezien ingebouwde methodes worden uitgevoerd in antwoord op acties, moet u, om de juiste ingebouwde methode te kunnen kiezen, eerst bepalen wat het object moet doen en wanneer.

4. Test de applicatie en breng verbeteringen aan.

   Met Paradox kunt u een applicatie gemakkelijk stukje voor stukje ontwikkelen. U hoeft niet de code voor de hele applicatie te schrijven voordat u het formulier kunt starten en kunt zien of alles werkt zoals u had verwacht.

***Opmerking*** Voor deze zelfstudie is de eerste fase al voor u uitgevoerd: de tabellen zijn gemaakt en voorzien van gegevens. Stel uw werkdirectory in op de directory met uw voorbeeldbestanden (meestal C:\PDOXWIN\VOORBD), zodat u toegang kunt krijgen tot deze tabellen en volg daarna de voorbeelden in deze zelfstudie.

Als u veranderingen hebt aangebracht in de voorbeeldtabellen, moet u deze opnieuw installeren voordat u met de zelfstudie begint. Als u dat niet doet, is het mogelijk dat u niet de verwachte resultaten krijgt van de voorbeelden in dit handboek. Als u de voorbeeldtabellen opnieuw wilt installeren, start u INSTALL en selecteert u alleen de optie 'Voorbeeldtabellen installeren'. Als u de voorbeeldtabellen via een netwerk gebruikt, moet u de systeembeheerder raadplegen.

# Formulieren maken

In deze paragraaf wordt eerst uitgelegd hoe u een één-record formulier maakt. Dat wil zeggen, een formulier waarin één record van een tabel tegelijk wordt weergegeven. In de volgende lessen wordt uitgelegd hoe u de volgende taken uitvoert:

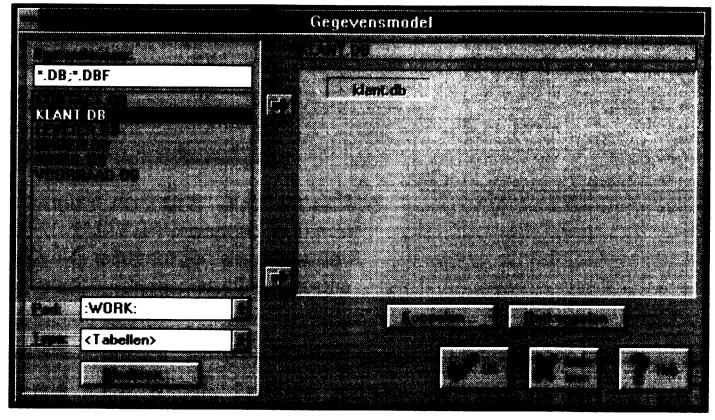
- Een knop programmeren zodat deze een handeling uitvoert
- Op een handeling reageren
- Naar waarden zoeken

Deze lessen beginnen met het maken van een formulie. Zodra u het formulier hebt gemaakt en opgeslagen, kunt u ditt bij alle volgende lessen in dit hoofdstuk gebruiken.

## Voorbeeld 3-1 Het formulier *NwKlant* maken

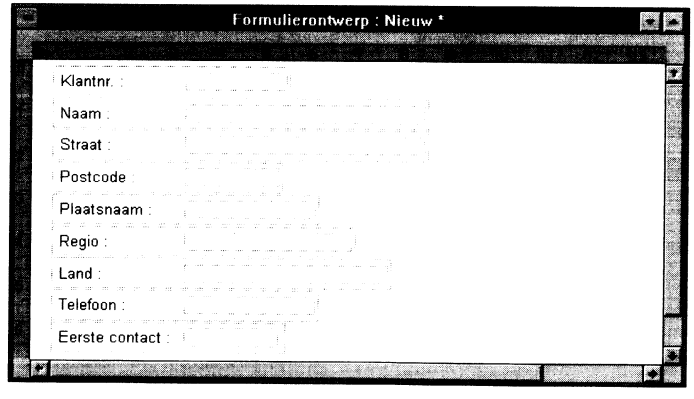
1. Kies 'BestandlNieuwlFormulier' om het dialoogvenster 'Gegevensmodel' te openen. Zorg ervoor dat u in de directory VOORBD werkt. (Zie Hoofdstuk 4 van *Aan de slag* voor meer informatie.)

2. Voeg de tabel *Klant* toe aan het gegevensmodel, zoals in de volgende afbeelding wordt getoond:



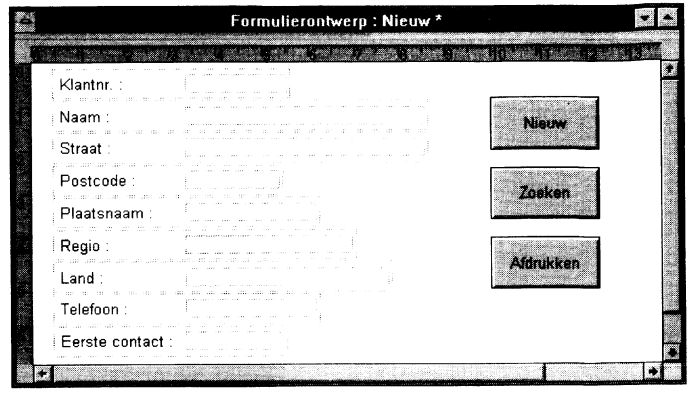

3. Kies 'OK' om dit gegevensmodel te accepteren en het dialoogvenster 'Layout ontwerpen' te openen.

4. Kies 'OK' om de standaard-layout te accepteren en het nieuwe formulier (zie hieronder) in een ontwerpvenster weer te geven.




5. Voeg vervolgens met het knop-hulpmiddel drie knoppen toe. Plaats de knoppen op het formulier en geef deze de labels *Nieuw*, *Zoeken* en *Afdrukken*, zoals in de volgende afbeelding:

Het label van een knop is gewoon een tekstvak. Als u de tekst wilt veranderen, selecteert u het label en wijzigt u de tekst.




6. Kies ten slotte 'Bestand|Opslaan' om het formulier op te slaan. Geef het formulier de naam NWKLANT.FSL. In deze lessen wordt naar dit formulier verwezen als het formulier *NwKlant*. U kunt het formulier *NwKlant* als uitgangspunt gebruiken voor de volgende lessen.

# Een knop programmeren zodat deze een handeling uitvoert

In deze les koppelt u code aan een knop om een bepaalde handeling te initiëren: een nieuw record invoegen.

## Voorbeeld 3-2 Code aan een knop koppelen

1. Inspecteer de knop met het label *Nieuw* en verander de naam in *nwKnop*. Als u de naam van een object wilt veranderen, inspecteert u het object en klikt u op de standaardnaam (het eerste item in het objectmenu). Het dialoogvenster 'Objectnaam' verschijnt dan. In dit dialoogvenster typt u de nieuwe naam.

2. Inspecteer *nwKnop* nogmaals om het menu te openen en kies 'Methodes' om het methodevenster te openen.

3. Kies **pushButton** en kies vervolgens 'OK' om een Editor-venster te openen voor de ingebouwde **pushButton**-methode.

4. Voeg code aan de **pushButton**-methode toe, zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   action(DataInsertRecord)
endmethod
```

5. Controleer de syntaxis en corrigeer eventuele fouten.

6. Start het formulier.

7. Klik op de knop 'Gegevens bewerken' (of kies 'Formulier|Gegevens bewerken' of druk op *F9*) om de bewerkmodus te activeren.
8. Klik op *nwKnop* om een nieuw, leeg record in te voegen. (In een latere les voegt u meer code toe aan de knop, zodat deze nuttiger wordt.)
9. U kunt gegevens in dit record invoeren of 'Record|Verwijderen' kiezen (of op *Ctrl-Del* drukken) om het record te verwijderen.

10. Keer terug naar het formulierontwerpvenster en sla het formulier op.

## Werking

Als deze **pushButton**-methode wordt uitgevoerd, heeft de instructie **action(DataInsertRecord)** exact hetzelfde effect als wanneer u 'Record | Invoegen' kiest of op *Ins* drukt. Met andere woorden: deze knop voegt geen nieuwe mogelijkheden toe. De procedure illustreert echter wel een belangrijk punt: een formulier reageert op dezelfde manier op een **action**-instructie van ObjectPAL als op een handeling van de gebruiker. Als u ObjectPAL leert, kan het zelfs handig zijn het formulier te beschouwen als een "vertaler" van gebruikers-handelingen in **action**-instructies.

De elementaire **action**-instructie bestaat uit twee delen:

- De methodenaam, **action**, die verwijst naar de **action**-methode die is gedefinieerd voor UIObjecten.
- Een ObjectPAL handelingsconstante (bijvoorbeeld DataInsertRecord) die aangeeft welke handeling moet worden uitgevoerd.

***Belangrijk*** De **action**-instructie is de basismethode om een handeling te initiëren vanuit ObjectPAL. De **action**-methode wordt gecombineerd met een constante die aangeeft wat er moet worden gedaan.

Tabel 3-1 geeft een overzicht van de handelingsconstanten die beschikbaar zijn op het beginnersniveau. Als u de mogelijkheden van ObjectPAL wilt benutten, is het van groot belang dat u deze constanten begrijpt.

Tabel 3-1 Handelingsconstanten van het beginnersniveau van ObjectPAL

| Constante | Beschrijving |
|---|---|
| DataInsertRecord | Voegt een record in |
| DataDeleteRecord | Verwijdert een record |
| DataLockRecord | Vergrendelt een record |
| DataUnlockRecord | Ontgrendelt een record |
| DataPostRecord | Voert een record door in de database |
| DataCancelRecord | Maakt veranderingen in een record ongedaan |
| DataBegin | Gaat naar het eerste record in een tabel |

| Constante | Beschrijving |
|---|---|
| DataEnd | Gaat naar het laatste record in een tabel |
| DataPriorRecord | Gaat naar het vorige record in een tabel |
| DataNextRecord | Gaat naar het volgende record in een tabel |
| DataBeginEdit | Start de bewerkmodus |
| DataEndEdit | Beëindigt de bewerkmodus |
| DataArriveRecord | Wijst naar een nieuw of gewijzigd record (wordt alleen gebruikt om op de handeling te reageren, niet om deze te initiëren) |
| FieldForward | Gaat naar het volgende object in de tab-volgorde |
| FieldBackward | Gaat naar het vorige object in de tab-volgorde |

# Reageren op een handeling

Het is niet voldoende om handelingen te initiëren. U moet ook besturen hoe objecten reageren. Zoals in de vorige lessen is uitgelegd, heeft elk object op een formulier ingebouwde methodes die meestal aan uw behoeften voldoen. Soms wilt u echter iets anders of iets meer. In deze les wordt uitgelegd hoe u bepaalt hoe een object op een bepaalde handeling reageert.

In deze les leert u de tab-volgorde te besturen. De tab-volgorde bepaalt naar welk object de cursor gaat als u op *Tab* drukt. Eerst moet u een indruk krijgen van de manier waarop een formulier zich standaard gedraagt.

### Voorbeeld 3-3 Standaardwerking van de Tab-toets

1. Start het formulier *NwKlant*. De cursor (markering) bevindt zich in het eerste veldobject, *Klantnr*.
2. Druk op *Tab*. De cursor gaat naar het veldobject *Naam*.
3. Druk nog een paar keer op *Tab* en kijk hoe de cursor van veldobject naar veldobject gaat.

Als u een formulier start en ermee werkt, genereert u handelingen. Op *Tab* drukken vormt daarop geen uitzondering. Het resultaat is een handeling. De constante die aangeeft om welke handeling het gaat, is FieldForward, zoals u in Tabel 3-1 ziet. (Als u op *Shift-Tab* drukt om terug te gaan, genereert u de handeling FieldBackward.)

De standaard tab-volgorde begint met het hoogste veldobject en werkt naar beneden. Stel echter dat u, nadat u het nummer en de naam van de klant hebt ingevoerd, meteen naar *Telefoon* wilt gaan en de andere veldobjecten wilt overslaan. Als u het telefoonnummer van de klant hebt ingevoerd, wilt u terugkeren naar *Straat* en de

adresgegevens invoeren. Met ObjectPAL is dit eenvoudig, als u twee belangrijke principes begrijpt:

- Elk object op een formulier (inclusief het formulier zelf) heeft een ingebouwde **action**-methode die wordt uitgevoerd in reactie op handelingen die worden gegenereerd door de gebruiker, door Paradox of door ObjectPAL.
- Als u de reactie van een object wilt wijzigen, koppelt u code aan de ingebouwde **action**-methode van het object.

InVoorbeeld 3-4 worden deze principes toegepast om te bepalen hoe het veldobject *Naam* reageert als u op *Tab* drukt.

### Voorbeeld 3-4 Tab-volgorde bepalen

1. Klik op de knop 'Ontwerpen' (of druk op *F8*) om terug te keren naar het ontwerpvenster.
2. Inspecteer *Naam* en kies 'Methodes' om het methodevenster te openen.
3. Kies **action** en kies vervolgens 'OK' om een Editor-venster te openen voor de ingebouwde **action**-methode.

***Snelle manier*** U kunt een Editor-venster ook openen door op een methode te dubbelklikken.

4. Bewerk de methode zodat deze er als volgt uitziet:

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = FieldForward then
      disableDefault
      Telefoon.moveTo()
   endIf
endmethod
```

5. Sluit dit Editor-venster en sla de veranderingen op als daarom wordt gevraagd.
6. Inspecteer *Telefoon* en kies 'Methodes' om het methodevenster te openen.
7. Kies **action** en kies vervolgens 'OK' om een Editor-venster te openen voor de ingebouwde **action**-methode.
8. Bewerk de methode zodat deze er als volgt uitziet:

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = FieldForward then
      disableDefault
      Straat.moveTo()
   endIf
endmethod
```

9. Controleer de syntaxis en corrigeer eventuele fouten.

10. Start het formulier en druk vervolgens tweemaal op **Tab** (of druk tweemaal op **Enter**; dit heeft hetzelfde effect). Controleer of u inderdaad naar *Telefoon* gaat.
11. Druk nogmaals op **Tab** en controleer of u inderdaad naar *Straat* gaat.

12. Schakel over naar het formulierontwerpvenster (**F8**) en sla het formulier op.

## Werking

De methodes die u in dit voorbeeld hebt aangepast, bevatten nu zes coderegels: twee regels worden standaard toegevoegd en vier regels vormen eigen code. In deze paragraaf wordt alleen eigen code besproken, te beginnen met de code die is gekoppeld aan het veldobject *Naam*.

*Regel 2* De regel met eigen code na de standaard methode-kopregel luidt

```
if eventInfo.id() = FieldForward then
```

Technisch gesproken wordt met deze regel getest of de waarde die door **eventInfo.id** wordt teruggegeven, gelijk is aan de waarde van de handelingsconstante FieldForward. Omdat dit een hele mond vol is, wordt hieronder dezelfde informatie stap voor stap herhaald:

- **if** markeert het begin van een **if...endIf**-blok. Met dit blok kunt u instructies alleen laten uitvoeren als aan bepaalde voorwaarden wordt voldaan.
- De **id**-methode werkt met de variabele *eventInfo* en geeft een waarde terug die de handeling identificeert. Onthoud dat *eventInfo* informatie bevat over de actie die de ingebouwde methode heeft geactiveerd. In dit geval is de actie een handeling en bevat *eventInfo* informatie die de handeling identificeert. De **id**-methode haalt deze informatie op.
- De teruggegeven waarde wordt vergeleken met de waarde van de handelingsconstante FieldForward. Alle ObjectPAL-constanten bevatten voorgedefinieerde waarden.
- Als de twee waarden hetzelfde zijn, wordt de volgende coderegel uitgevoerd. De volgende regel wordt dus alleen uitgevoerd als de teruggegeven waarde gelijk is aan FieldForward. Als de teruggegeven waarde anders is (bijvoorbeeld DataInsertRecord), wordt de rest van dit **if...endIf** -blok niet uitgevoerd. In plaats daarvan wordt dan de standaardcode uitgevoerd en wordt de methode beëindigd.

*Regel 3* De derde regel van de methode luidt

```
disableDefault
```

Met **disableDefault** wordt voorkomen dat de ingebouwde code van deze methode wordt uitgevoerd. Zoals u hebt gezien, houdt het

standaardgedrag in dit geval in dat de cursor naar het veldobject *Straat* wordt verplaatst. Als u **disableDefault** aanroept, wordt dat voorkomen.

*Regel 4* De vierde regel van de methode luidt

```
Telefoon.moveTo()
```

Zoals u weet, is *Telefoon* de naam van een veldobject op dit formulier. **moveTo** is een methode die de cursor naar een object verplaatst. Met deze instructie wordt de cursor dus naar *Telefoon* verplaatst.

*Regel 5* De vijfde regel van de methode luidt

```
endIf
```

**endIf** markeert het einde van een **if...endIf**-blok.

De code die is gekoppeld aan *Telefoon*, is exact hetzelfde, met uitzondering van de volgende instructie, die de cursor naar het veld *Straat* verplaatst:

```
Straat.moveTo()
```

***Belangrijk*** Dit eenvoudige voorbeeld illustreert de algemene methode voor het reageren op handelingen in ObjectPAL: koppel code aan de ingebouwde **action**-methode van een object, roep **eventInfo.id** aan om de handeling te identificeren en gebruik handelingsconstanten om te testen op de handeling of handelingen waarop u wilt reageren.

U kunt handelingen initiëren en erop reageren met behulp van dezelfde handelingsconstanten. De volgende code gebruikt bijvoorbeeld de handelingsconstante DataPostRecord om een handeling te initiëren.

```
method pushButton(var eventInfo Event)
   action(DataPostRecord)
endmethod
```

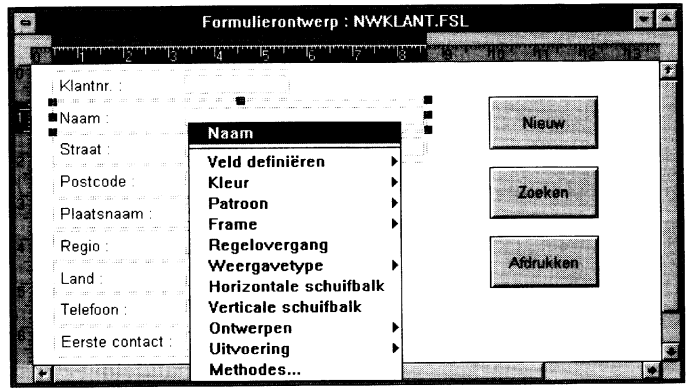
In de volgende code wordt dezelfde handelingsconstante gebruikt om op een handeling te reageren: als de handeling DataPostRecord is, toont de code een bericht in een dialoogvenster. Vervolgens wordt de standaardcode uitgevoerd en wordt het record doorgevoerd. Als het een andere handeling betreft, gaat Paradox naar het einde van de methode en wordt de standaardcode uitgevoerd.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataPostRecord then
      msgInfo("NB", "Klaar om huidige record op te slaan.")
   endIf
endmethod
```

# Handelingen en kenmerken

In deze les ziet u, net als in de vorige les, hoe u op een handeling reageert. Daarnaast wordt in deze les duidelijk gemaakt hoe u met objectkenmerken werkt en hoe u de kenmerken van één object manipuleert op basis van de kenmerken van een ander object. Elk ontwerpobject heeft bepaalde eigenschappen en attributen, die in Paradox *kenmerken* worden genoemd. Als u een object inspecteert, worden in het objectmenu allerlei kenmerken weergegeven, zoals in Afbeelding 3-1. Met ObjectPAL hebt u toegang tot al deze kenmerken en nog een groot aantal andere.

Afbeelding 3-1 Kenmerken van het veldobject *Naam*




In Voorbeeld 3-5 ziet u hoe u met de handelingsconstante DataArriveRecord reageert op een handeling, die op verschillende manieren kan zijn geïnitieerd. Een DataArriveRecord-handeling vindt plaats als het formulier een nieuw of ander record "bereikt" (weergeeft). Als u bijvoorbeeld naar het volgende of vorige record gaat, wordt een DataArriveRecord-handeling geïnitieerd. Hetzelfde gebeurt als u een record invoegt, verwijdert of bewerkt. De handeling DataArriveRecord is vaak handig. (U kunt alleen reageren op een DataArriveRecord-handeling en deze niet initiëren.)

Stel dat u met klantgegevens werkt en de namen wilt markeren van "oude" klanten, dat wil zeggen, klanten met wie u contact hebt opgenomen voor 1 januari 1991. In het volgende voorbeeld ziet u hoe u dit doet.

Gebruik tijdens deze les het formulier *NwKlant* dat u eerder in dit hoofdstuk hebt gemaakt (open het in een ontwerpvenster).

### Voorbeeld 3-5 Reageren op een handeling

1. Inspecteer het veldobject *Naam* en kies 'Methodes' om het methodevenster te openen.
2. Kies **action** en kies vervolgens 'OK' om een Editor-venster te openen voor de ingebouwde **action**-methode.

***Opmerking*** Als u de vorige lessen hebt doorgenomen, is er misschien al eigen code gekoppeld. U kunt deze code verwijderen of er voorlopig commentaar van maken. Als u van code commentaar wilt maken, plaatst u de code tussen accolades { }.

3. Bewerk de methode zodat deze er als volgt uitziet:

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataArriveRecord then
      if Eerste_contact.Value < Date("1-01-91") then
         Self.Color = Green
      else
         Self.Color = White
      endIf
   endIf
endmethod
```

4. Controleer de syntaxis en corrigeer eventuele fouten.

5. Start het formulier.
6. Verplaats het invoegpunt naar het veldobject *Naam* en blader vervolgens door de records.
7. Let op de kleur van het veldobject *Naam*. *De kleur moet groen zijn als de waarde van het veldobject Eerste_contact* kleiner is dan 1-01-91; anders moet de kleur wit zijn.

## Werking

Deze methode wordt uitgevoerd als het veldobject *Naam* op een handeling reageert (en werkt alleen als u het invoegpunt naar *Naam*) verplaatst. Bij alle handelingen, *behalve* DataArriveRecord, wordt alleen de ingebouwde code uitgevoerd en gedraagt *Naam* zich als elk ander veldobject. Als de handeling echter DataArriveRecord is, wordt de eigen code uitgevoerd en doet *Naam* iets speciaals.

*Regel 2* De tweede regel van de methode luidt

```
if eventInfo.id() = DataArriveRecord then
```

Deze instructie kijkt of de handeling DataArriveRecord is. Dat is namelijk de handeling waarin u bent geïnteresseerd.

*Regel 3* De derde regel luidt

```
if Eerste_contact.Value < Date("1-01-91") then
```

Deze instructie leest het kenmerk 'Value' van het veldobject *Eerste_contact* en kijkt of deze waarde kleiner (vroeger) is dan 1-01-91.

**Belangrijk** In de tabel *Klant* bevat de veldnaam *Eerste contact* een spatie, maar op dit formulier bevat de naam van het veldobject *Eerste_contact* een onderstrepingsteken. De regel luidt als volgt: namen van velden in tabellen mogen spaties en punten bevatten, maar namen van objecten op een formulier mogen dat niet.

*Werken met de kenmerken van een object*

Als u met de kenmerken van een object werkt, gebruikt u de volgende syntaxis:

```
objectNaam.kenmerkNaam
```

Vervang *objectNaam* door de naam van een object (*Eerste_contact* in dit voorbeeld) en vervang *kenmerkNaam* door de naam van een kenmerk, 'Waarde' ('Value') in dit voorbeeld).

Met het kenmerk 'Value' kunt u de waarde van een object lezen en schrijven. De volgende instructie plaatst bijvoorbeeld de waarde 1234 in het veldobject *Klantnr.*, alsof u deze zelf typt.

```
Klantnr_.Value = 1234
```

**Belangrijk** ObjectPAL gebruikt een speciale syntaxis bij datumwaarden. De aanhalingstekens in dit voorbeeld maken een uitdrukking van de datum en zorgen ervoor dat ObjectPAL 1-01-91 als een datum behandelt (anders wordt het symbool - als een rekenoperator behandeld):

```
Date("1-01-91")
```

ObjectPAL behandelt de waarde van het veldobject *Eerste_contact* als een datum, omdat het als een datumveld is gedefinieerd in de bijbehorende tabel (KLANT.DB).

*Regel 4*

De vierde regel luidt

```
Self.Color = Green
```

*De variabele Self*

Deze instructie gebruikt de algemene syntaxis voor kenmerken, *objectNaam.kenmerkNaam*. De kenmerknaam is 'Color', maar wat is *Self*? *Self* is een ingebouwde objectvariabele die verwijst naar het object dat de huidige code uitvoert.

In deze instructie is het object dat de code uitvoert, het veldobject *Naam* (u hebt het invoegpunt in stap 6 van deze oefening naar *Naam* verplaatst). *Self* verwijst dus naar *Naam*.

*Self* is in dit geval een gemakkelijke oplossing. In gecompliceerde applicaties is *Self* een belangrijk element in algemene code, omdat er objecten mee kunnen worden bewerkt, die niet bij naam worden genoemd.

# Samenvatting

In deze lessen zijn enkele krachtige technieken behandeld. U hebt de volgende zaken geleerd:

- Handelingen initiëren met de UIObject-methode **action**
- Reageren op handelingen met de ingebouwde **action**-methode
- De werking van handelingsconstanten
- Voorkomen dat standaardcode wordt uitgevoerd
- Standaardcode vervangen door uw eigen code
- Werken met objectkenmerken
- Werken met de objectvariabele *Self*

In de volgende lessen worden deze technieken verder besproken en leert u nog meer controle uit te oefenen over een applicatie.

HOOFDSTUK 4

# Invoer en uitvoer

De lessen in dit hoofdstuk laten zien hoe informatie van de gebruiker wordt verkregen, hoe informatie aan de gebruiker wordt getoond en hoe deze informatie verder wordt verwerkt.

In dit hoofdstuk komen de volgende onderwerpen aan de orde:

- Een snelle manier om gebruikersinvoer te verkrijgen
- Waarden zoeken in een tabel
- Een record invoegen en een unieke sleutelwaarde genereren
- Een rapport afdrukken

Aan de hand van de voorbeelden in deze lessen wordt uitgelegd hoe u code toevoegt aan het eerder gemaakte formulier *NwKlant*.

## Een snelle manier om gebruikersinvoer te verkrijgen

In deze les leert u hoe u met de **view**-methode een dialoogvenster opent waarin een gebruiker een waarde kan invoeren. Bovendien leert u hoe u variabelen declareert en gebruikt in ObjectPAL. Dit voorbeeld draagt niet veel bij aan de totstandkoming van een applicatie, maar kan u wel op praktische ideeën brengen.

### Voorbeeld 4-1 Een dialoogvenster openen met **view**

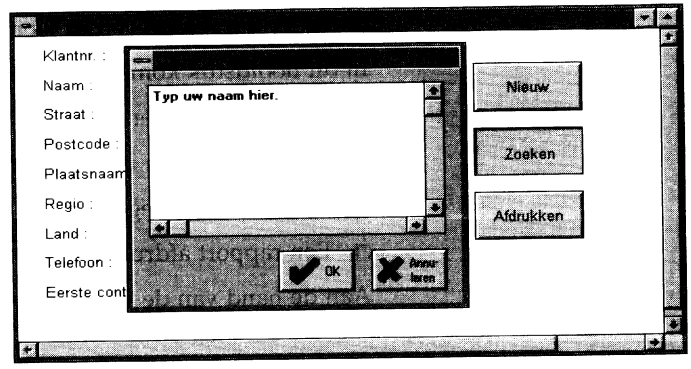
1. Open het formulier *NwKlant* in het ontwerpvenster.
2. Inspecteer de knop met de naam *Zoeken* en verander de naam in *zoekKnop*.
3. Selecteer *zoekKnop* en druk op ***Ctrl-Spatiebalk*** om het methodevenster te openen.
4. Dubbelklik op **pushButton** om deze methode te selecteren. Er wordt een Editor-venster geopend voor de **pushButton**-methode.

5. Bewerk de methode, zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   var
      gebrInvoer String
   endVar

   gebrInvoer = "Typ uw naam hier."
   gebrInvoer.view("Hoe heet u?")

   message("Hallo ", gebrInvoer)
   sleep(1000)
endmethod
```

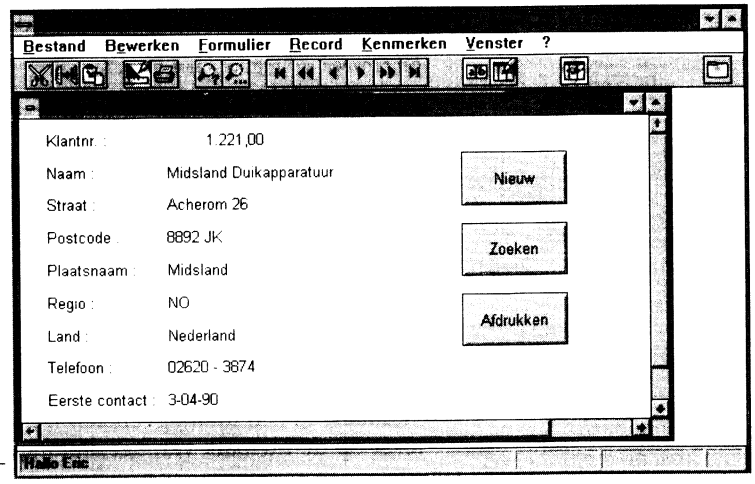
6. Controleer de syntaxis en corrigeer eventuele fouten.

7. Start het formulier en klik op de knop *Zoeken*. Er verschijnt een dialoogvenster waarin u wordt gevraagd uw naam te typen:




8. Typ uw naam in het dialoogvenster en druk op ***Enter*** of klik op 'OK'.

9. Op de statusbalk in de linkerbenedenhoek van het formuliervenster verschijnt een bericht.




10. Keer terug naar het formulierontwerpvenster en kies 'Bestand|Opslaan' om het formulier op te slaan.

## Werking

In dit voorbeeld wordt met een paar coderegels veel bereikt. Hieronder volgt een uitleg van de eigen coderegels.

*Regel 2* De regel met eigen code na de ingebouwde methode-kopregel bestaat uit één sleutelwoord:

```
var
```

Het sleutelwoord **var** geeft het begin aan van een blok waarin variabelen worden gedeclareerd. (Variabelen worden gedeclareerd door middel van een naam en een gegevenstype.)

*Regel 3* De derde regel luidt

```
gebrInvoer String
```

Deze code declareert een variabele met de naam *gebrInvoer* van het gegevenstype String. Nu kunt u de variabele *gebrInvoer* in deze methode gebruiken om een tekenreeks op te slaan.

***Opmerking*** In ObjectPAL hoeft u geen variabelen te definiëren, maar het heeft voordelen om het wel te doen. Code wordt dan namelijk sneller uitgevoerd, bevat minder snel syntaxisfouten en is gemakkelijker te lezen en te onderhouden.

*Regel 4* De vierde regel luidt

```
endVar
```

Het sleutelwoord **endVar** markeert het einde van het blok waarin variabelen worden gedeclareerd.

*Regel 6* De zesde regel van de methode luidt

```
gebrInvoer = "Typ uw naam hier."
```

Deze instructie wijst een waarde toe aan *gebrInvoer*. Deze instructie zorgt er dus voor dat de tekenreeks "Typ uw naam hier." in de variabele wordt opgeslagen. Met ObjectPAL moet u een waarde eerst aan een variabele toekennen, voordat u deze kunt gebruiken in een andere instructie of uitdrukking.

*Regel 7* De zevende regel luidt

```
gebrInvoer.view("Hoe heet u?")
```

Als deze instructie wordt uitgevoerd, wordt de methode **view** gebruikt op de variabele *gebrInvoer*. **View** toont de waarde van de variabele in een dialoogvenster. De reeks "Hoe heet u?" bepaalt welke tekst op de titelbalk van het dialoogvenster verschijnt.

**View** zorgt niet alleen dat de waarde van de variabele in een dialoogvenster wordt weergegeven. Als u het dialoogvenster namelijk sluit, wijst **view** de weergegeven waarde weer toe aan de variabele. Deze nieuwe waarde wordt gebruikt in de volgende regel met eigen code.

Het **view**-dialoogvenster kan zowel numerieke invoer als tekenreeksen verwerken. De volgende code vraagt u bijvoorbeeld een creditcard-nummer te typen:

```
method pushButton(var eventInfo Event)
   var
      cardNummer Number
   endVar

   cardNummer = 0       ; wijs een tijdelijke waarde toe
   cardNummer.view("Typ het creditcard-nummer.")
   message(cardNummer) ; toon de nieuwe waarde
   sleep(1000)
endmethod
```

*Regel 9* De negende regel luidt

```
message("Hallo ", gebrInvoer)
```

Deze instructie roept de systeemprocedure **message** aan met twee argumenten: de vaste reeks "Hallo" en de variabele *gebrInvoer*. In dit voorbeeld is de waarde van *gebrInvoer* de naam die u hebt ingevoerd in het **view**-dialoogvenster. Als u bijvoorbeeld de naam Anja hebt ingevoerd, geeft deze **message**-instructie de tekst **Hallo Anja** weer op de statusbalk.

*Regel 10* De laatste regel met eigen code luidt

```
sleep(1000)
```

Een **sleep**-instructie zorgt ervoor dat het systeem een bepaald aantal milliseconden "wacht"; daarna gaat het systeem verder. In dit

voorbeeld zijn 1000 milliseconden (één seconde) opgegeven om u de kans te geven het bericht te lezen.

# Waarden zoeken

In deze les leert u hoe u een waarde vraagt aan de gebruiker, hoe u in een tabel een record zoekt dat die waarde bevat en hoe u dat record aan de gebruiker toont. U gebruikt een **view**-dialoogvenster om invoer van de gebruiker te krijgen (zie de vorige les). Vervolgens zoekt u de waarde met behulp van de **locate**-methode en laat u de weergave over aan het formulier.

Gebruik voor het volgende voorbeeld het formulier *NwKlant*, dat u in vorige lessen hebt gemaakt.

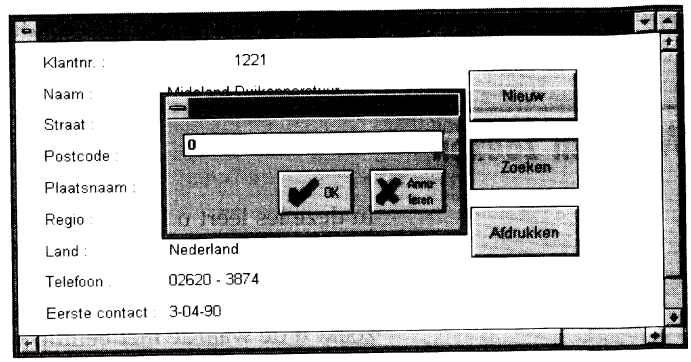
## Voorbeeld 4-2 Zoeken op basis van invoer van de gebruiker

1. Het formulier *NwKlant* moet geopend zijn in een ontwerpvenster.
2. Inspecteer *Zoeken* en kies 'Methodes' om het methodevenster te openen.
3. Dubbelklik op **pushButton** om een Editor-venster te openen voor de ingebouwde **pushButton**-methode.
4. Bewerk de methode, zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   var
      klantNum Number
   endVar

   klantNum = 0
   klantNum.view("Voer een klantnummer in:")

   if klantNum <> 0 then
      if not Klantnr_.locate("Klantnr.", klantNum) then
         beep()
         message("Niet gevonden ", klantNum)
         sleep(1000)
      endIf
   endIf
endmethod
```

5. Controleer de syntaxis en corrigeer eventuele fouten.

6. Start het formulier en klik op *Zoeken*. Het onderstaande dialoogvenster verschijnt. Hierin wordt u gevraagd een klantnummer te typen.




7. Typ **1560** in het dialoogvenster en druk op ***Enter*** of klik op 'OK' om het dialoogvenster te sluiten.
8. Dit nummer bevindt zich in de tabel *Klant* en dus geeft het formulier het record weer.
9. Klik nogmaals op *Zoeken* om een **view**-dialoogvenster te openen.
10. Typ **1** in het dialoogvenster en druk op ***Enter*** of klik op 'OK' om het dialoogvenster te sluiten.
11. Dit nummer bevindt zich niet in de tabel *Klant*. Paradox geeft daarom een geluidssignaal en het formulier toont een bericht op de statusbalk.
12. Keer terug naar het formulierontwerpvenster en kies 'BestandlOpslaan' om het formulier op te slaan.

## Werking

In dit voorbeeld worden verschillende elementen gebruikt die reeds in de vorige les zijn uitgelegd. Alleen de nieuwe elementen worden hier gedetailleerd behandeld.

*Regels 2 tot en met 4* De regels twee tot en met vier luiden

```
var
   klantNum Number
endVar
```

Deze code definieert de variabele met de naam *klantNum* van het type Number.

*Regels 6 en 7* De volgende twee regels luiden

```
klantNum = 0
klantNum.view("Voer een klantnummer in:")
```

Deze code wijst de waarde 0 toe aan *klantNum*, geeft de waarde weer in een dialoogvenster en wacht totdat u een nieuwe waarde invoert en het dialoogvenster sluit.

*Regel 9* De negende regel luidt

```
if klantNum <> 0 then
```

Deze code voert een eenvoudige test uit om te controleren of u een waarde in het dialoogvenster hebt ingevoerd. De vorige instructie kende de waarde 0 toe aan *klantNum*. Deze instructie test de waarde van *klantNum*. Als de waarde niet 0 is, wordt de volgende coderegel uitgevoerd. Als de waarde wel 0 is, gaat Paradox naar het einde van de methode en wordt de standaardcode uitgevoerd.

*Regel 10* De tiende regel luidt

```
if not Klantnr_.locate("Klantnr.", klantNum) then
```

Analyseer deze instructie stap voor stap om goed te begrijpen wat er gebeurt. Kijk eerst naar **Klantnr_.locate("'Klantnr.'", klantNum)**. *Klantnr_* is de naam van een veldobject in dit formulier en is verbonden met de tabel *Klant*. **locate** is de naam van een methode. *Klantnr.* is de naam van een veld in de tabel *Klant*. *klantNum* is een variabele.

Met deze code wordt in feite gezegd "Zoek in de tabel *Klant* een record waarin de waarde van het veld *'Klantnr.'* overeenkomt met de waarde van de variabele *klantNum*". Met andere woorden: u gebruikt het object *Klantnr_* om de tabel op te geven, de **locate**-methode om de zoekactie op te geven, de veldnaam 'Klantnr.' om aan te geven naar welk veld wordt gezocht en de variabele *klantNum* om de waarde op te geven waarnaar wordt gezocht.

***Belangrijk*** De veldnaam *Klantnr.* bevat een punt, maar de naam van het veldobject *Klantnr_* bevat een onderstrepingsteken. Veldnamen in tabellen kunnen punten bevatten, maar objectnamen op een formulier kunnen dat niet.

Een elementaire **locate**-instructie bestaat uit de volgende onderdelen:

- Een objectnaam. Dit is de naam van een veldobject dat met een tabel is verbonden. Dit is de gemakkelijkste manier om aan te geven welke tabel wordt gezocht.
- De methodenaam **locate**.
- Een veldnaam.
- Een waarde waarnaar wordt gezocht.

De **locate**-methode zoekt vanaf het eerste record in de tabel naar een waarde die precies overeenkomt met de opgegeven waarde. Als deze wordt gevonden, geeft het formulier dat record weer. U hoeft verder niets te doen. Als de zoekactie mislukt, geeft het formulier het huidige record weer.

De rest van deze instructie bevat de bekende **if...then**-constructie met een nieuw element: **not**. Zoals u in de vorige lessen hebt gezien, wordt met een **if...then**-instructie getest of aan een voorwaarde wordt

voldaan. Als u **not** toevoegt, test u of niet aan een voorwaarde wordt voldaan. De volledige instructie betekent dus "Als er geen waarde te vinden is die overeenkomt met *klantNum*, worden de volgende coderegels uitgevoerd. Anders wordt naar het einde van de methode gesprongen en wordt de standaardcode uitgevoerd."

*Regels 11 tot en met 13* De volgende regels delen u mee dat een zoekactie niet is gelukt:

```
beep()
message("Niet gevonden ", klantNum)
sleep(1000)
```

De **beep**-instructie zorgt ervoor dat het systeem een geluidssignaal geeft om de gebruiker te waarschuwen. De **message**-instructie en de **sleep**-instructie worden in Voorbeeld 4-1 uitgelegd.

ObjectPAL kent nog krachtigere versies van **locate**, andere methodes waarmee u naar een tekenpatroon kunt zoeken in plaats van naar een identieke waarde en methodes die voorwaarts en achterwaarts zoeken in een tabel.

***Belangrijk*** ObjectPAL kent geen methodes voor zoeken en vervangen. U kunt voor een dergelijke bewerking het best gebruik maken van een *query*. Raadpleeg het *Handboek* voor meer informatie.

# Een record invoegen en een unieke sleutelwaarde genereren

Een van de voordelen van ObjectPAL is dat de taal u de mogelijkheid biedt taken te automatiseren. Stel dat uw werk bestaat uit het invoeren van bestellingen. Als u het formulier alleen interactief gebruikt (dus zonder gebruik te maken van ObjectPAL), moet u de volgende stappen uitvoeren om een bestelling in te voeren:

1. Kies 'Formulier | Gegevens bewerken'.
2. Kies 'Record | Invoegen'.
3. Typ een nieuwe, unieke waarde in het veldobject *'Klantnr_'*.

De laatste stap is de moeilijkste, omdat u alle klantnummers die de tabel al bevat, moet kennen om elke nieuwe klant een uniek nummer te kunnen geven.

In deze les leert u hoe u ObjectPAL gebruikt om slechts eenmaal met de muis te hoeven klikken om deze drie stappen uit te voeren. Net als in de vorige lessen wordt het formulier *NwKlant* gebruikt.

## Voorbeeld 4-3 Een nieuw record invoegen

1. Inspecteer *nwKnop* (de knop met het label *Nieuw*) en kies 'Methodes' om het methodevenster te openen.

2. Dubbelklik op **pushButton** om een Editor-venster te openen voor de **pushButton**-methode.

***Opmerking*** Als u de vorige lessen hebt doorgenomen, is het mogelijk dat er al eigen code aan de methode is gekoppeld. U kunt deze code verwijderen of er tijdelijk commentaar van maken. Als u van code commentaar wilt maken, plaatst u de code tussen accolades { }.

3. Bewerk de methode, zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   var
      nwKlantNum Number
      klantTbl Table
   endVar

   action(DataBeginEdit)
   action(DataInsertRecord)

   klantTbl.attach("KLANT.DB")
   nwKlantNum = klantTbl.cMax("Klantnr.") + 1
   Klantnr_.Value = nwKlantNum
   action(DataPostRecord)
endmethod
```

4. Controleer de syntaxis en corrigeer eventuele fouten.

5. Start het formulier en klik op *nwKnop*. Het formulier activeert de bewerkmodus, plaatst een record in de tabel en een nieuwe waarde in het veldobject *Klantnr_*. U hebt nu een nieuw record dat u kunt bewerken.

6. Keer terug naar het formulierontwerpvenster en kies 'Bestand|Opslaan' om het formulier op te slaan.

## Werking

De code in dit voorbeeld is verdeeld in drie blokken. In de eerste twee blokken worden elementen gebruikt die in de vorige lessen zijn uitgelegd. In het derde blok worden nieuwe elementen geïntroduceerd. Deze worden hier uitgebreid besproken.

*Blok 1* Het eerste blok eigen code luidt

```
var
   nwKlantNum Number
   klantTbl Table
endVar
```

In dit blok worden twee variabelen gedefinieerd: *nwKlantNum* van het type Number en *klantTbl* van het type Table. Deze variabelen werken enigszins verschillend. Een Number-variabele bevat een numerieke waarde, terwijl een Table-variabele voor een *handle* zorgt. Deze handle kunt u in uw code gebruiken om naar een tabel te verwijzen en deze te manipuleren.

*Blok 2* Het tweede blok eigen code luidt

```
action(DataBeginEdit)
action(DataInsertRecord)
```

Dit blok initieert twee handelingen. De eerste instructie, **action (DataBeginEdit)**, activeert de bewerkmodus voor het formulier. Als het formulier zich al in de bewerkmodus bevindt, wordt deze instructie genegeerd. De tweede instructie, **action(DataInsertRecord)**, opent een nieuw, leeg record, zoals in het vorige hoofdstuk is uitgelegd.

*Blok 3* Het derde blok eigen code luidt

```
klantTbl.attach("KLANT")
nwKlantNum = klantTbl.cMax("Klantnr.") + 1
Klantnr_.Value = nwKlantNum
action(DataPostRecord)
```

De eerste instructie in dit blok bestaat uit de volgende elementen: de Table-variabele *klantTbl*, de methodenaam **attach** en de bestandsnaam van de tabel *Klant*, KLANT.DB.

***Opmerking*** ObjectPAL zoekt bestanden standaard in uw werkdirectory. U kunt een andere plaats opgeven door de volledige padnaam of een alias toe te voegen aan de bestandsnaam.

Zoals eerder is gezegd, zorgt een Table-variabele voor een handle voor de tabel. Deze **attach**-instructie koppelt de Table-variabele *klantTbl* aan de Paradox-tabel KLANT.DB. Als u nu *klantTbl* gebruikt in deze methode, verwijst u naar de tabel *Klant*.

De tweede instructie wijst een waarde toe aan de Number-variabele *nwKlantNum*. De Table-variabele *klantTbl*, de methode **cMax** en het veld 'Klantnr.' worden gebruikt om deze waarde te bepalen. De **cMax**-methode geeft de maximale waarde uit een opgegeven kolom van een tabel terug. In dit voorbeeld wordt de **cMax**-methode gebruikt op de variabele *klantTbl*, die is verbonden met de tabel *Klant*. Het veld 'Klantnr.' van elk record in de tabel wordt bekeken en de hoogste waarde wordt teruggegeven. Als u bij die waarde één optelt, krijgt u gegarandeerd een unieke waarde. Stel dat **cMax** de waarde 4456 teruggeeft. Dat betekent dat 4456 momenteel het hoogste ordernummer in de tabel is en dat 4456 + 1, oftewel 4457, dus uniek moet zijn. Deze waarde wordt toegekend aan *nwKlantNum*. Het is van belang dat de waarde uniek is, omdat deze wordt gebruikt in het sleutelveld *Klantnr.*, dat per definitie een unieke waarde vereist.

De derde instructie wijst de waarde van *nwKlantNum* toe aan het veldobject *Klantnr_*. Het kenmerk 'Value' (besproken in Voorbeeld 3-5) wordt gebruikt om een waarde op te geven die in een veldobject moet worden opgeslagen en weergegeven.

De laatste instructie in dit blok initieert een DataPostRecord-handeling om het nieuwe record (inclusief het nieuwe klantnummer) door te voeren in de tabel *Klant*.

***Opmerking*** De methode in dit voorbeeld is bedoeld voor single-user-applicaties. In Hoofdstuk 18 worden methodes behandeld die u in een multi-user-applicatie kunt gebruiken.

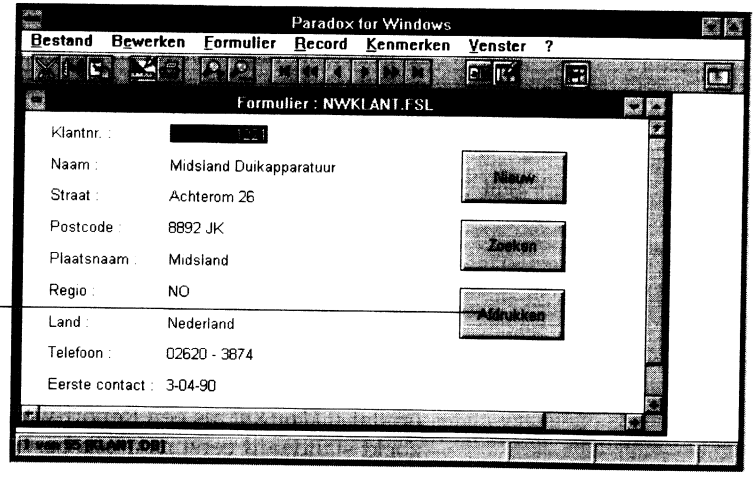
# Een rapport afdrukken

In deze les leert u de basismethode om een vooraf ontworpen rapport af te drukken. Dat wil zeggen, een rapport dat al is ontworpen en op schijf is opgeslagen. Als u de vorige lessen hebt doorgenomen, kunt u het formulier *NwKlant* voor deze les gebruiken.

### Voorbeeld 4-4 Een vooraf ontworpen rapport afdrukken

De code in dit voorbeeld is gekoppeld aan de knop *Afdrukken* van het formulier *NwKlant*.

U maakt een eenvoudig rapport op basis van de tabel *Klant*. Geef dit rapport de naam KLANT.RSL.




1. Inspecteer de knop *Afdrukken* en verander de naam in *afdrukKnop*.
2. Inspecteer *afdrukKnop* nogmaals en kies 'Methodes' om het methodevenster te openen.
3. Dubbelklik op **pushButton** om een Editor-venster te openen voor de **pushButton**-methode.
4. Bewerk de **pushButton**-methode, zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   var
     klantRpt Report
   endVar

   if klantRpt.open("KLANT") then
      klantRpt.print()
   else
      msgInfo("Probleem", "Kan het rapport niet openen.")
   endIf
endmethod
```

5. Controleer de syntaxis en corrigeer eventuele fouten.

6. Start het formulier en klik op *afdrukKnop*. Paradox laadt het rapport vanaf de schijf en opent het dialoogvenster 'Bestand afdrukken'. Kies in dit dialoogvenster afdrukinstellingen, zoals een paginabereik en de manier waarop overloop moet worden behandeld. Klik vervolgens op 'OK' om het rapport naar de printer te sturen.

7. Sluit het rapport.

8. Keer terug naar het formulierontwerpvenster en kies 'Bestand|Opslaan' om het formulier op te slaan.

## Werking

De eigen code in deze methode is verdeeld in twee blokken.

*Blok 1* In het eerste blok wordt een Report-variabele met de naam *klantRpt* gedefinieerd. Evenals een Table-variabele (besproken in Voorbeeld 4-3) fungeert een Report-variabele als een handle voor een rapportbestand dat op de schijf is opgeslagen.

*Blok 2* Het tweede blok luidt

```
if klantRpt.open("KLANT") then
   klantRpt.print()
else
   msgInfo("Probleem", "Kan het rapport niet openen.")
endIf
```

De eerste instructie in dit blok probeert het rapportbestand van de schijf te lezen. Paradox zoekt automatisch naar een rapportbestand, omdat u *klantRpt* als een Report-variabele hebt gedeclareerd. Paradox zoekt standaard eerst naar een bestand met de naam KLANT.RSL. Als dit bestand niet wordt gevonden, wordt naar KLANT.RDL gezocht. Als een van deze twee bestanden wordt gevonden, probeert Paradox het rapport te openen. Als dit lukt, wordt de variabele *klantRpt* een handle naar het rapport en zorgt de tweede instructie, **klantRpt.print()**, ervoor dat het rapport wordt afgedrukt. Als Paradox het rapportbestand om de een of andere reden niet kan openen, opent de code een dialoogvenster dat u op de hoogte brengt van het probleem. Dit is de manier waarop ObjectPAL normaal gesproken op fouten reageert.

# Samenvatting

In de lessen van dit hoofdstuk hebt u geleerd hoe u

- Een **view**-dialoogvenster gebruikt om invoer van de gebruiker te krijgen
- Een waarde in een tabel zoekt met **locate**
- De gebruiker informatie geeft met **beep**, **message** en **sleep**
- De bewerkmodus voor een formulier activeert
- Een nieuw, leeg record invoegt
- De grootste waarde in de kolom van een tabel zoekt
- Een waarde aan een veld toewijst
- Een rapport afdrukt

HOOFDSTUK 5

# Gegevensinvoer valideren

De lessen in dit hoofdstuk beschrijven manieren om ervoor te zorgen dat gebruikers geldige gegevens invoeren. De volgende onderwerpen worden besproken:

- De validiteitscontroles gebruiken die zijn ingebouwd in Paradox
- ObjectPAL gebruiken om de validiteit van velden te controleren
- ObjectPAL gebruiken om de validiteit van records te controleren

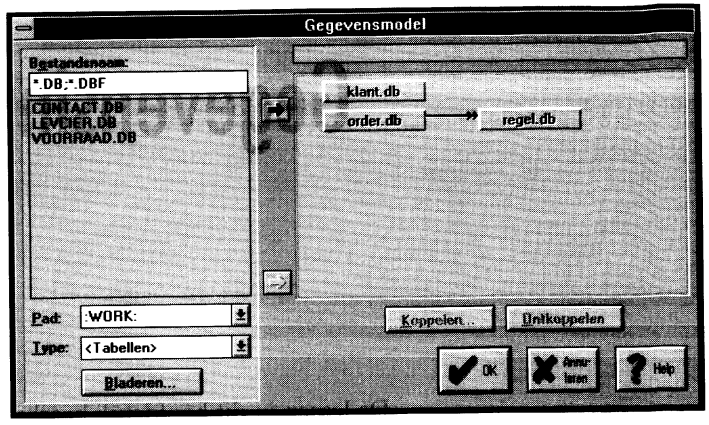
In de eerste lessen wordt een nieuw formulier gebruikt. In Voorbeeld 5-1 wordt beschreven hoe u dit formulier maakt.

## Een multi-tabel formulier maken

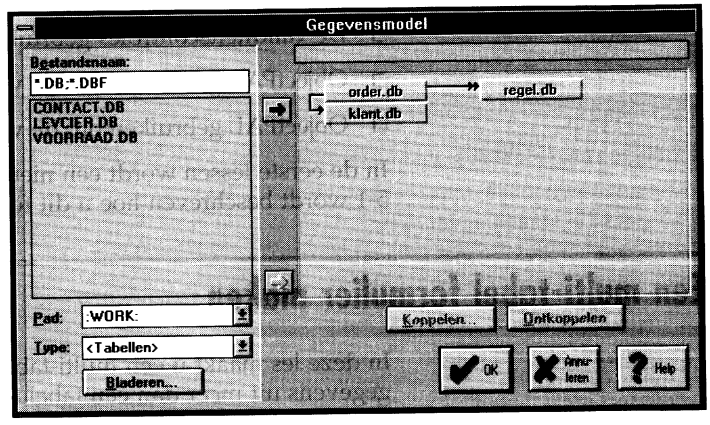
In deze les maakt u een multi-tabel formulier (een formulier dat gegevens uit meer dan één tabel toont).

### Voorbeeld 5-1 Een multi-tabel formulier maken

1. Kies 'BestandINieuwIFormulier' om het dialoogvenster 'Gegevensmodel' te openen.
2. Voeg de tabellen *Klant, Regel* en *Order* toe aan het gegevensmodel.
3. In dit formulier is *Order* de hoofdtabel. Koppel *Order* op de onderstaande manier aan *Regel*:




4. Koppel *Order* vervolgens aan *Klant* (zie de onderstaande afbeelding).

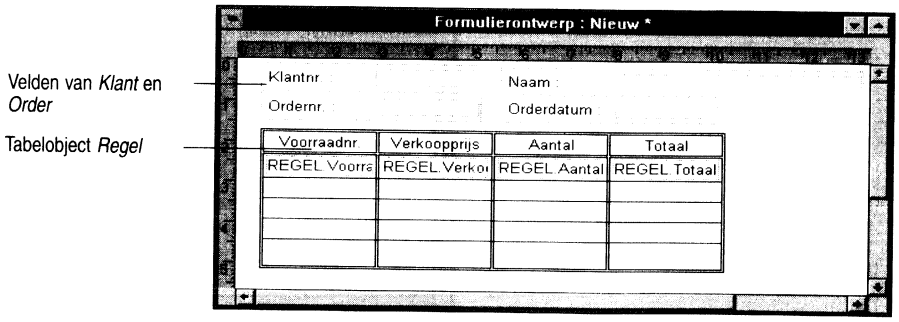



5. Kies 'OK' om dit gegevensmodel te gebruiken. Het dialoogvenster 'Layout ontwerpen' verschijnt.

6. In het dialoogvenster 'Layout ontwerpen' selecteert u de optie 'Velden voor tabellen' in het paneel 'Object-layout'. U kunt de objecten in dit formulier dan gemakkelijker zien.

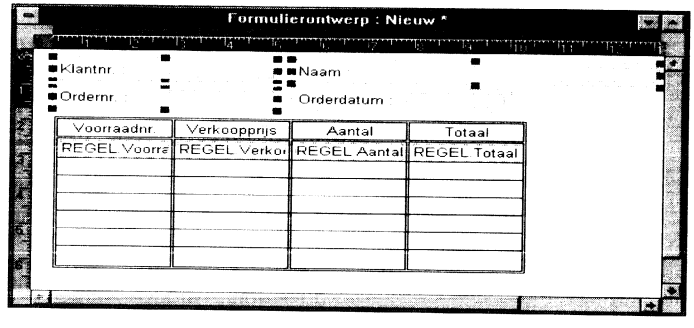
7. Kies 'OK' om de layout te accepteren en het nieuwe formulier in een ontwerpvenster weer te geven.

8. Deze applicatie gebruikt niet alle velden uit de tabellen. De applicatie gebruikt het tabelframe dat is verbonden met *Regel*, het veldobject *Naam* dat is verbonden met *Klant*, en de veldobjecten *Ordernr.*, *Klantnr.* en *Orderdatum*, die alle zijn verbonden met de tabel *Order*. Verwijder de andere veldobjecten en rangschik de overblijvende objecten zo dat uw formulier er als volgt uitziet:



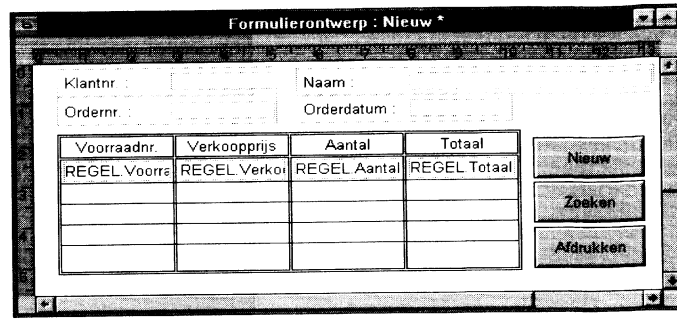

*Tabstop*

9. Houd ***Shift*** ingedrukt en klik op de volgende veldobjecten: *Klantnr.*, *Ordernr.* en *Naam*. Als u alle drie de veldobjecten hebt geselecteerd (zie de onderstaande afbeelding), inspecteert u een van deze objecten om de kenmerken te bekijken. Kies 'Uitvoering|Tabstop' om het kenmerk 'Tabstop' te deselecteren. Op deze manier stelt u het kenmerk 'Tabstop' voor alle drie de velden tegelijk in.




Doordat u het kenmerk 'Tabstop' deselecteert, zorgt u ervoor dat de gebruiker de cursor niet meer naar een van deze velden kan verplaatsen. Zo krijgt u meer controle over de applicatie, omdat de gebruiker geen verkeerde gegevens in deze velden kan plaatsen. Verander het kenmerk 'Tabstop' van *Orderdatum* niet. In dit veldobject gaat u in de volgende les gegevens invoeren.

10. Gebruik vervolgens het knop-hulpmiddel om drie knoppen toe te voegen. Plaats de knoppen op het formulier en geef deze de namen *Nieuw*, *Zoeken* en *Afdrukken* (zie de onderstaande afbeelding).



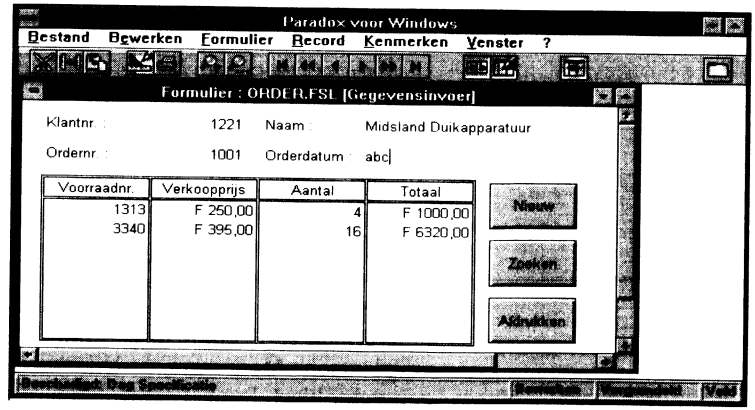

11. Kies 'Bestand|Opslaan' om het formulier te bewaren. Noem het ORDER.FSL. In de volgende lessen wordt dit formulier *Order* genoemd.

# Ingebouwde validiteitscontroles

De ingebouwde validiteitscontroles van Paradox zijn zeer krachtig. U kunt deze controles interactief leren gebruiken, zoals wordt beschreven in het *Handboek*. U hoeft dan beduidend minder code te schrijven om een goede applicatie te maken. Hier volgt een zeer eenvoudig voorbeeld van het gebruik van validiteitscontrole.

## Voorbeeld 5-2 Ingebouwde validiteitscontrole

1. Toen de tabel *Order* werd gemaakt, is *Orderdatum* gedefinieerd als een datumveld. Paradox weigert daarom elke waarde die geen geldige datum tussen 1 januari 100 en 31 december 9999 oplevert. Start het formulier *Order* om deze validiteitscontrole te zien werken.
2. Druk op ***F9*** om de bewerkmodus te activeren.
3. Ga naar het veldobject *Orderdatum*.
4. Typ **abc**.
5. Dit is geen geldige datum, dus verschijnt er een bericht op de statusbalk van het formulier, zoals in de volgende afbeelding. Dit gebeurt omdat de ingebouwde validiteitscontrole van Paradox een fout heeft ontdekt.




6. Kies 'Bewerken|Ongedaan maken' om de oorspronkelijke (geldige) datum terug te halen.

# Validiteitscontroles toevoegen met ObjectPAL

Hoewel de ingebouwde validiteitscontroles van Paradox krachtig zijn, hebt u soms meer controle nodig. Het kan bijvoorbeeld onvoldoende zijn dat de waarde van het veldobject *Orderdatum* een geldige datum is.

Stel dat u wilt voorkomen dat de gebruiker een datum typt die in de toekomst ligt. In het volgende voorbeeld ziet u hoe u een strengere validiteitscontrole kunt uitvoeren.

## Voorbeeld 5-3 Validiteitscontrole met ObjectPAL

1. Ga terug naar het ontwerpvenster.
2. Inspecteer het veldobject *Orderdatum* en kies 'Methodes' om het methodevenster te openen.
3. Dubbelklik op **canDepart** om een Editor-venster te openen voor de ingebouwde **canDepart**-methode.
4. Bewerk de methode zodat deze er als volgt uitziet:

```
method canDepart(var eventInfo MoveEvent)
   if Self.Value > today() then
      eventInfo.setErrorCode(CanNotDepart)
      message("Orderdatum kan niet later zijn dan de datum van vandaag.")
      sleep(1000)
   endIf
endmethod
```

5. Controleer de syntaxis en corrigeer eventuele fouten.

6. Start het formulier en activeer de bewerkmodus.
7. Verplaats de cursor naar het veld *Orderdatum*, typ een toekomstige datum en druk op ***Enter***.
8. Kijk naar de melding op de statusbalk.
9. Kies 'BewerkenlOngedaan maken' om de oorspronkelijke datum weer op te roepen.

## Werking

Elk object heeft een ingebouwde methode met de naam **canDepart**. In feite vraagt **canDepart** toestemming om de cursor uit het object te verplaatsen. Als u eigen code koppelt aan de **canDepart**-methode van een veldobject, kunt u controleren of de gebruiker een geldige waarde in het veldobject heeft getypt voordat het veld verlaten kan worden.

*Regel 2* De tweede regel van de methode luidt

```
if Self.Value > today() then
```

De belangrijkste elementen in deze instructie zijn de objectvariabele *Self*, het kenmerk 'Value' en de procedure **today** uit de run-time bibliotheek. In dit voorbeeld voert het veldobject *Orderdatum* de code uit, zodat *Self* verwijst naar *Orderdatum*.

De procedure **today** geeft de huidige datum terug, volgens de interne klok van uw computer.

De volledige instructie vergelijkt de waarde van *Orderdatum* met de huidige datum. Als *Orderdatum* groter is, wordt de volgende regel eigen code uitgevoerd; anders wordt de uitvoering van de methode beëindigd.

*Regel 3* De derde regel luidt

```
eventInfo.setErrorCode(CanNotDepart)
```

Deze instructie gebruikt de methode **setErrorCode**, die is gedefinieerd voor het type MoveEvent, om informatie op te slaan in de variabele *eventInfo*. In deze instructie gebruikt **setErrorCode** de ObjectPAL-constante CanNotDepart om een fout aan te geven. In dit geval wordt de fout veroorzaakt door een datum in de toekomst en slaat de constante CanNotDepart informatie op in *eventInfo*. Als informatie is opgeslagen in *eventInfo*, is deze beschikbaar voor Paradox en kan Paradox erop reageren. In dit geval luidt het antwoord dat de cursor het veldobject niet mag verlaten voordat de gebruiker een waarde opgeeft die voldoet aan de opgegeven criteria.

***Belangrijk*** De normale manier om een foutvoorwaarde aan te kondigen, is **setErrorCode** en een foutconstante te gebruiken. Zo wordt informatie doorgegeven aan de variabele *eventInfo*, waarop Paradox vervolgens

kan reageren. Raadpleeg de online Help voor een compleet overzicht van de ObjectPAL-constanten.

*Regels 4 en 5* De vierde en de vijfde regel luiden

```
message("Orderdatum kan niet later zijn dan de datum van vandaag.")
sleep(1000)
```

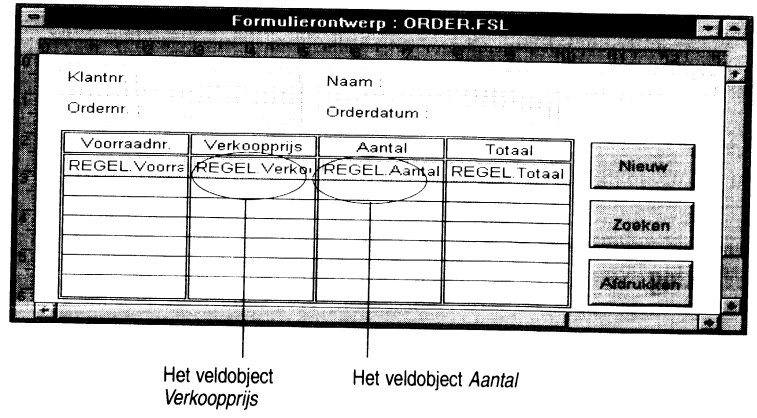
De **message**-instructie toont een foutmelding op de statusbalk en de **sleep**-instructie zorgt voor een pauze, zodat u tijd hebt om het bericht te lezen.

# Waarden verstrekken

Soms kunt u het beste zelf geldige waarden verstrekken. In het vorige hoofdstuk werd in Voorbeeld 4-3 een uniek ordernummer gemaakt dat in een veldobject werd geplaatst, zodat de gebruiker dat niet meer hoeft te doen. U kunt het de gebruiker ook gemakkelijker maken door ObjectPAL te gebruiken om, waar mogelijk, berekeningen uit te voeren. In deze les ziet u één manier om dit te doen.

Zoals u in Afbeelding 5-1 en in Voorbeeld 5-4 ziet, bevat het formulier *Order* een tabelframe dat verbonden is met *Regel*. Dit tabelframe bevat records die bestaan uit de volgende veldobjecten: *Voorraadnr.*, *Verkoopprijs*, *Aantal* en *Totaal*. De waarde van *Totaal* is voor elk besteld artikel het produkt van de waarden van *Verkoopprijs* en *Aantal*.

Afbeelding 5-1 Het formulier *Order*



De volgende stappen laten zien hoe u deze berekening uitvoert met ObjectPAL.

### Voorbeeld 5-4 Berekeningen uitvoeren

1. Keer terug naar het ontwerpvenster.
2. Inspecteer *Verkoopprijs* en kies 'Methodes' om het methodevenster te openen.
3. Dubbelklik op **changeValue** om een Editor-venster te openen voor de ingebouwde **changeValue**-methode.
4. Bewerk de methode zodat deze er als volgt uitziet:

```
method changeValue(var eventInfo ValueEvent)
   doDefault
   if not Aantal.isBlank() then
      Totaal.Value = Self.Value * Aantal.Value
   endIf
endmethod
```

5. Inspecteer *Aantal* en kies 'Methodes' om het methodevenster te openen.
6. Dubbelklik op **changeValue** om een Editor-venster te openen voor de ingebouwde **changeValue**-methode.
7. Bewerk de methode zodat deze er als volgt uitziet:

```
method changeValue(var eventInfo ValueEvent)
   doDefault
   if not Verkoopprijs.isBlank() then
      Totaal.Value = Self.Value * Verkoopprijs.Value
   endIf
endmethod
```

8. Controleer de syntaxis en verbeter eventuele fouten.

9. Start het formulier en activeer de bewerkmodus.
10. Typ verschillende waarden in *Verkoopprijs* en *Aantal*, zodat u kunt zien dat ObjectPAL de berekening uitvoert.

11. Keer terug naar het formulierontwerpvenster en sla het formulier op.

## Werking

In dit voorbeeld wordt code gebruikt die is gekoppeld aan twee objecten: *Verkoopprijs* en *Aantal*. De code is gekoppeld aan de ingebouwde **changeValue**-methode van deze objecten.

Veldobjecten hebben een ingebouwde methode met de naam **changeValue**. De methode **changeValue** vraagt in feite toestemming om de waarde van het veldobject door te voeren in de onderliggende tabel. Als u eigen code koppelt aan de ingebouwde **changeValue**-methode van een veldobject, kunt u bepalen welke reactie optreedt als de gebruiker de waarde van het veldobject verandert. De code

voor deze objecten is bijna identiek. Hieronder wordt de code voor *Verkoopprijs* besproken en daarna worden de verschillen met de code voor *Aantal* bekeken.

*Regel 2* De tweede regel van de methode luidt

```
doDefault
```

Een aanroep van **doDefault** voert de standaardcode voor deze ingebouwde methode onmiddellijk uit; zonder deze aanroep zou de standaardcode pas aan het einde van de methode worden uitgevoerd. In het geval van **changeValue** geeft de aanroep van **doDefault** u toegang tot de bijgewerkte waarde van het veldobject. Als u bijvoorbeeld een verkoopprijs voor een nieuw record invoert, krijgt ObjectPAL pas toegang tot die waarde als de standaardcode van **changeValue** wordt uitgevoerd. Stel dat u een bestaande bestelling bewerkt en dat u de verkoopprijs verandert van F 12,95 in F 14,99. ObjectPAL zal dan de oude prijs gebruiken, totdat de standaardcode van **changeValue** wordt uitgevoerd.

U kunt dit zien als u de volgende code koppelt aan de **changeValue**-methode van *Verkoopprijs*:

```
method changeValue(var eventInfo ValueEvent)
   msgInfo("Vóór aanroep van doDefault", Self.Value)
   doDefault
   msgInfo("Na aanroep van doDefault", Self.Value)
endmethod
```

Start het formulier, activeer de bewerkmodus en typ een waarde in *Verkoopprijs*. Er verschijnen dialoogvensters waarin u de waarde ziet van *Verkoopprijs* vóór en na de aanroep van **doDefault**. Als u een andere waarde invoert, ziet u eerst de oude waarde in een dialoogvenster en daarna de gewijzigde waarde.

*Regel 3* De derde regel roept de **isBlank**-methode uit de runtime bibliotheek aan. Deze methode werkt hier op *Aantal*. **isBlank** levert True op als het veldobject leeg is. Als het veldobject wel een waarde bevat, levert **isBlank** False op. In dit voorbeeld heeft het geen zin de berekening uit te voeren als *Aantal* leeg is. In dat geval gaat de uitvoering dus naar het einde van de methode.

*Regel 4* De vierde regel luidt

```
Totaal.Value = Self.Value * Aantal.Value
```

Deze code voert de berekening uit. De waarde van *Verkoopprijs* (vertegenwoordigd door de objectvariabele *Self*) wordt vermenigvuldigd met de waarde van *Aantal* en het resultaat wordt toegekend aan het kenmerk 'Value' van *Totaal*.

De code die is gekoppeld aan de ingebouwde **changeValue**-methode van *Aantal*, lijkt een spiegelbeeld van de zojuist besproken code, met de volgende verschillen:

- De code kijkt eerst of *Verkoopprijs* leeg is voordat de berekening wordt uitgevoerd.
- De objectvariabele *Self* verwijst naar *Aantal*. Zoals u weet, verwijst *Self* naar het object waar de code aan is gekoppeld die op het moment wordt uitgevoerd.

# Sleutelinbreuken afhandelen

In de vorige lessen hebt u gezien hoe u een validiteitscontrole uitvoert op velden (dat wil zeggen, hoe u de waarden van de afzonderlijke velden controleert). In deze les worden validiteitscontroles op recordniveau behandeld. U leert met name hoe u sleutelinbreuken opvangt.

Deze les bestaat uit twee onderdelen. In het eerste deel leert u aan de hand van het formulier *NwKlant* hoe u sleutelinbreuken opvangt in een één-record formulier. Vervolgens leert u aan de hand van het formulier *Order* hoe u sleutelinbreuken opvangt in een multi-record formulier.

## Eén-record formulieren

In de vorige lessen hebt u gewerkt met afzonderlijke objecten die zich op een formulier bevonden. In deze les wordt het formulier als ontwerpobject geïntroduceerd. U leert hoe het formulier de objecten beheert die het insluit en hoe het acties en handelingen overziet.

### Voorbeeld 5-5 Een formulier als beheerder

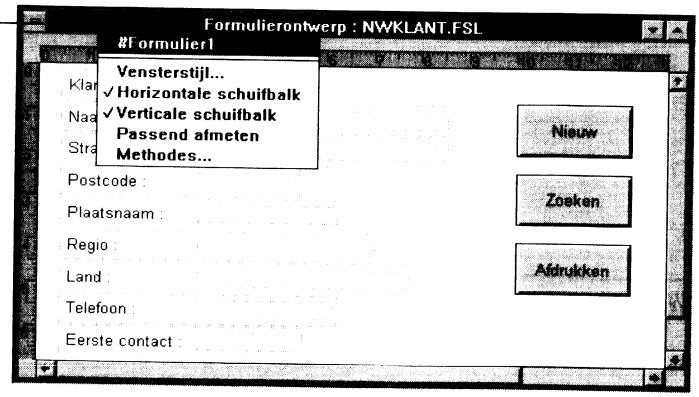
Open *NwKlant* in het ontwerpvenster en ga als volgt te werk:

1. Kies 'KenmerkenlFormulierlMethodes' om het dialoogvenster 'Methodes' te openen. In dit dialoogvenster verschijnen de ingebouwde methodes van het formulier.

***Snellere manier***

U kunt een formulier inspecteren door met de rechtermuisknop te klikken op de titelbalk van het formuliervenster of door op **Esc** te drukken totdat alle andere objecten gedeselecteerd zijn, en vervolgens op *F6* te drukken.

Klik met de rechtermuisknop op de titelbalk van het formuliervenster om een formulier te inspecteren




2. Dubbelklik op **action** om een Editor-venster te openen voor de ingebouwde **action**-methode van het formulier.

   Het venster van ingebouwde formuliermethodes bevat enige aanvullende standaardcode. Deze code is bestemd voor gevorderde ObjectPAL-gebruikers en wordt besproken in Hoofdstuk 12.

3. Bewerk de methode zodat deze er als volgt uitziet:

```
method action(var eventInfo ActionEvent)
   if eventInfo.isPreFilter() then
      ; deze code wordt uitgevoerd voor alle objecten op het formulier

   else
      ; deze code wordt alleen uitgevoerd voor het formulier zelf
      if eventInfo.id() = DataUnlockRecord or
         eventInfo.id() = DataPostRecord then
         doDefault
         if eventInfo.errorCode() = peKeyViol then
            msgInfo("Probleem", "Typ een ander klantnummer.")
         endIf
      endIf
   endIf
endmethod
```

4. Controleer de syntaxis en verbeter eventuele fouten.

5. Start het formulier en activeer de bewerkmodus (hierdoor vergrendelt u het record).

6. Bewerk het eerste record als volgt: verander de waarde van *Klantnr.* in **1351**. Omdat de tabel dit nummer al bevat, veroorzaakt u een sleutelinbreuk en treedt er een fout op als u de volgende stap uitvoert.

7. Druk op ***F9*** om de bewerkmodus te verlaten (en het record te ontgrendelen). Uw code reageert op de sleutelinbreuk: er verschijnt een dialoogvenster met het bericht dat u een ander klantnummer moet typen. Het formulier blijft in de bewerkmodus.

8. Kies 'OK' om het dialoogvenster te sluiten.

9. Druk op ***Ctrl-F5*** om het record door te voeren (zonder het te ontgrendelen).
10. Het dialoogvenster verschijnt weer. Kies 'OK' om het te sluiten.
11. Kies 'RecordIWijzigingen annuleren' (of druk op ***Alt-Backspace***) om de oorspronkelijke waarde van het veldobject te herstellen.

12. Keer terug naar het formulierontwerpvenster en kies 'BestandIOpslaan' om het formulier op te slaan.

## Werking

U hebt zojuist gezien hoe een formulier zaken beheert. Terwijl u met de objecten op het formulier werkt, genereert u acties en initieert u handelingen. Deze handelingen en acties gaan eerst naar het formulier en het formulier beslist wat ermee wordt gedaan. In dit voorbeeld test het formulier op twee specifieke handelingen: DataUnlockRecord en DataPostRecord. Als het formulier een van deze handelingen ontvangt, controleert het of er sprake is van een sleutelinbreuk. Als er zich een inbreuk heeft voorgedaan, wordt dit gemeld. De code die voor deze les van belang is, begint bij regel 7.

*Regel 7 and 8* Regel 7 en 8 luiden

```
if eventInfo.id() = DataUnlockRecord or
   eventInfo.id() = DataPostRecord then
```

Dit is eigenlijk één instructie die voor de leesbaarheid in twee regels is gesplitst. De instructie roept de **id**-methode aan om de handeling te identificeren. Als het gaat om een DataUnlockRecord- of DataPostRecord-handeling worden de volgende regels uitgevoerd om te controleren of er een sleutelinbreuk is opgetreden.

Er treedt een DataUnlockRecord-handeling op als u om de een of andere reden een record wilt ontgrendelen en doorvoeren: als u de bewerkmodus wilt verlaten, naar het volgende of het vorige record wilt gaan, een record wilt invoegen, een record wilt verwijderen enzovoort.

Er treedt een DataPostRecord-handeling op als u een record wilt doorvoeren in de onderliggende tabel en het record toch vergrendeld wilt houden.

*Regel 9* Regel 9 luidt

```
doDefault
```

Zoals u in de vorige les hebt gezien, voert **doDefault** de standaard-code van een ingebouwde methode uit. In dit voorbeeld voert **doDefault** de standaardcode voor het ontgrendelen of doorvoeren van een record uit. Als dit om de een of andere reden niet lukt, verschijnt er een foutmelding die u op de hoogte brengt.

*Regel 10* Regel 10 luidt

```
if eventInfo.errorCode() = peKeyViol then
```

Deze instructie maakt gebruik van de procedure **errorCode** om de foutcode te testen die is opgeslagen in de variabele *eventInfo*. De foutconstante peKeyViol van ObjectPAL wordt gebruikt om te zoeken naar een specifieke fout: een inbreuk op een sleutel. ObjectPAL kent niet alleen constanten voor handelingen, maar ook voor veel voorkomende foutvoorwaarden.

*Regel 11* Regel 11 luidt

```
msgInfo("Probleem", "Typ een ander klantnummer.")
```

Deze instructie opent een dialoogvenster met de mededeling dat u een ander klantnummer moet opgeven. Dit gebeurt omdat de tabel *Klant* slechts één sleutelveld heeft, namelijk 'Klantnr.'. Als er een inbreuk op een sleutel optreedt, moet de oorzaak dus een dubbel klantnummer zijn.

*Samenvatting* In dit voorbeeld hebt u gezien hoe u in een één-record formulier een inbreuk op een sleutel kunt ontdekken en is het formulier geïntroduceerd als een ontwerpobject dat acties en handelingen beheert voor de objecten die het insluit. Alles gaat eerst naar het formulier, zodat dit voor de andere objecten kan reageren op handelingen.

In dit voorbeeld hebt u gezien hoe u het volgende kunt doen:

- De procedure **errorCode** gebruiken om informatie over fouten te krijgen
- Foutconstanten gebruiken om specifieke fouten te identificeren

## Multi-tabel formulieren

In Voorbeeld 5-6 ziet u hoe u sleutelinbreuken kunt opvangen in een multi-tabel formulier. Voor dit voorbeeld gebruikt u het formulier *Order*. Zoals u al in Voorbeeld 5-4 hebt gezien, sluit het formulier *Order* veldobjecten en een tabelframe in. In Voorbeeld 5-5 hebt u gezien hoe u sleutelinbreuken in een één-record formulier kunt opvangen. U gebruikt dezelfde methode om sleutelinbreuken op te vangen in de veldobjecten van een multi-tabel formulier.

Het kenmerk 'Tabstop' van de veldobjecten op het formulier *Order* staat echter uit, zodat de gebruiker de cursor daar niet naar kan verplaatsen. Daardoor kan de gebruiker deze veldobjecten niet bewerken en kunnen deze velden nooit een inbreuk op een sleutel veroorzaken.

De veldobjecten in het tabelframe kunnen echter wel worden bewerkt, dus dit formulier heeft een mechanisme nodig om sleutelinbreuken op dat niveau te verwerken.

### Het principe dichterbij-is-beter

U kunt de methode uit het eerste deel van deze les gebruiken: koppel code aan de ingebouwde **action**-methode van het formulier en controleer op fouten na een DataUnlockRecord of een DataPostRecord-handeling. In het algemeen is het echter verstandig code *zo dicht mogelijk* bij het object te plaatsen waaraan de code is verbonden. Als u dit doet, is uw code modulair, objectgeoriënteerd en gemakkelijk te onderhouden en opnieuw te gebruiken.

In een één-record formulier zoals *NwKlant* kunt u sleutelinbreuken alleen op het formulier zelf afhandelen. In een multi-tabel formulier zoals *Order* hebt u twee mogelijkheden: u kunt de code koppelen aan het formulier of aan het tabelframe. Als u het formulier ziet als de beheerder van alle objecten die het insluit, kunt u een tabelframe beschouwen als een onderbeheerder: het tabelframe beheert alleen de veldobjecten en de recordobjecten die het bevat. Het formulier ziet alle acties en handelingen voor alle objecten op het formulier; het tabelframe ziet alleen de acties en de handelingen voor de objecten die het bevat. Als u het principe "dichterbij-is-beter" toepast, koppelt u de code aan het tabelframe.

#### Voorbeeld 5-6 Sleutelinbreuken afhandelen in een multi-tabel formulier

1. Open het formulier *Order* in een ontwerpvenster en inspecteer het tabelframe *REGEL*.
2. Kies 'Methodes' en vervolgens **action** om een Editor-venster te openen voor de ingebouwde **action**-methode van het tabelframe.
3. Bewerk de methode zodat deze er als volgt uitziet:

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataUnlockRecord or
      eventInfo.id() = DataPostRecord then
      doDefault
      if eventInfo.errorCode() = peKeyViol then
         msgInfo("Probleem", "Typ een ander voorraadnummer.")
      endIf
   endIf
endmethod
```

4. Controleer de syntaxis en verbeter eventuele fouten.

5. Start het formulier en activeer de bewerkmodus (het record wordt vergrendeld).
6. Verplaats de cursor naar het tweede record in het tabelframe. Verander de waarde van *Voorraadnr.* in **1313**. Het voorraadnummer voor het tweede record is nu hetzelfde als het voorraadnummer voor het eerste record. Zo veroorzaakt u een inbreuk op een sleutel.
7. Druk op ***F11*** om de cursor naar het eerste record te verplaatsen (en het tweede record te ontgrendelen). Er wordt een dialoogvenster geopend met de melding dat u een ander voorraadnummer moet typen. De cursor

gaat niet naar het volgende record. (Als u op ***F9*** had gedrukt, zoals in het vorige voorbeeld, had u hetzelfde resultaat gezien.)

8. Druk op ***Enter*** om het dialoogvenster te sluiten.
9. Druk op ***Ctrl-F5*** om het record door te voeren zonder het te ontgrendelen.
10. Het dialoogvenster verschijnt weer. Druk op ***Enter*** om het te sluiten.
11. Druk op ***Alt-Backspace*** (of kies 'Record|Wijzigingen annuleren') om de oorspronkelijke waarde van het veldobject te herstellen.
12. Keer terug naar het formulierontwerpvenster en kies 'Bestand|Opslaan' om het formulier te bewaren.

***Werking***

Deze code is op twee uitzonderingen na identiek aan de code in Voorbeeld 5-5:

- De code bevat geen standaardtekst voor ingebouwde methodes op formulierniveau.
- In het dialoogvenster staat dat u een ander voorraadnummer moet opgeven in plaats van een ander klantnummer.

Verder werkt alles precies hetzelfde, maar wel op lokaal niveau, omdat de code is gekoppeld aan het tabelframe en niet aan het formulier.

# Samenvatting

In dit hoofdstuk zijn methodes behandeld om validiteitscontroles uit te voeren op recordniveau. In de lessen hebt u het volgende gezien:

- Reageren op handelingen die een inbreuk op een sleutel kunnen veroorzaken
- Inbreuken op een sleutel opvangen in één-record formulieren
- Inbreuken op een sleutel opvangen in multi-record formulieren
- Het formulier gebruiken voor het beheer van de objecten die het insluit
- Een tabelframe gebruiken als onderbeheerder van de veldobjecten en de recordobjecten die het insluit
- Het principe "dichterbij-is-beter" toepassen om te bepalen waar u uw code plaatst

HOOFDSTUK 6

# Andere formulieren besturen

In de lessen in dit hoofdstuk leert u hoe u ObjectPAL gebruikt om met een formulier een ander formulier te besturen. De voorbeelden laten zien hoe u een formulier als dialoogvenster gebruikt, maar u kunt deze kennis toepassen in elke formulier applicatie met meerdere formulieren. In deze lessen is het formulier *Order* het *aanroepend formulier*; dat wil zeggen dat het formulier *Order* een tweede formulier (het dialoogvenster) aanroept (opent). Het tweede formulier is in deze lessen het formulier *Klant*.

In Voorbeeld 6-1 en Voorbeeld 6-2 ziet u hoe u van een formulier een dialoogvenster maakt. In Voorbeeld 6-3 ziet u hoe u een dialoogvenster aanroept en beheert.

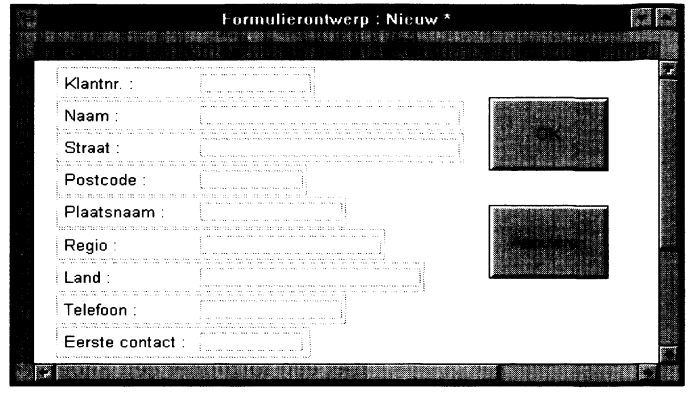
## Dialoogvensters ontwerpen

Een dialoogvenster is een formulier met een aantal speciale kenmerken. Als u een dialoogvenster wilt ontwerpen, ontwerpt u een formulier en stelt u vervolgens de benodigde kenmerken in, zoals u in de volgende voorbeelden ziet.

### Voorbeeld 6-1 Een dialoogvenster ontwerpen

1. Kies 'BestandINieuwIFormulier' om het dialoogvenster 'Gegevensmodel' te openen.
2. Kies KLANT.DB uit de lijst met tabellen en voeg de tabel toe aan het gegevensmodel.
3. Klik op 'OK' om het dialoogvenster 'Gegevensmodel' te sluiten. Het dialoogvenster 'Layout ontwerpen' wordt geopend.
4. Klik op 'OK' om de standaard-layout te accepteren.

5. Gebruik het knop-hulpmiddel om twee knoppen te plaatsen, zoals in de volgende afbeelding:

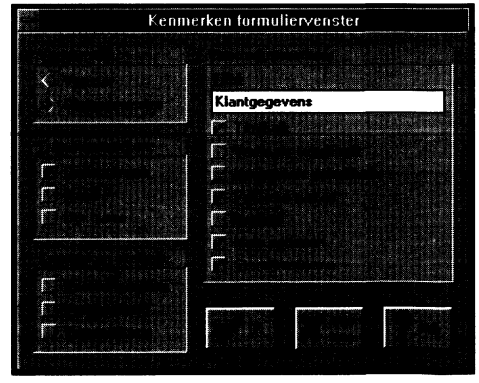



6. Geef één knop het label *OK* en verander de naam in *okKnop*.
7. Geef de andere knop het label *Annuleren* en verander de naam in *annuleerKnop*.
8. Sla het formulier op en noem het KLANT.FSL.

**Tot nu toe hebt u niets anders gedaan dan wanneer u een gewoon formulier ontwerpt. In Voorbeeld 6-2 ziet u hoe u de kenmerken van dit formulier instelt, zodat het lijkt op een dialoogvenster en zich ook zo gedraagt.**

### Voorbeeld 6-2 De kenmerken van een formulier instellen

1. Kies 'KenmerkenlFormulierlVensterstijl aanpassen' om het dialoogvenster 'Kenmerken formuliervenster' te openen (zie de volgende afbeelding).




2. Selecteer of deselecteer vakken, zodat het dialoogvenster er uitziet als het voorbeeld hierboven. Het is belangrijk dat u de kenmerken precies zo instelt, anders gedraagt het dialoogvenster zich niet goed.

3. Kies 'OK' om het dialoogvenster te sluiten. U ziet nog geen veranderingen op het formulier. Deze treden pas op als u het formulier start.
4. Kies 'BestandIOpslaan' om deze veranderingen te bewaren.

   Dat is alles. Het ontwerp van het dialoogvenster is nu klaar. De volgende stap is code te koppelen aan de knoppen.
5. Koppel de volgende code aan de ingebouwde **pushButton**-methode van *okKnop*.

```
method pushButton(var eventInfo Event)
   formReturn("OK")
endmethod
```

6. Koppel de volgende code aan de ingebouwde **pushButton**-methode van *annuleerKnop*.

```
method pushButton(var eventInfo Event)
   formReturn("Annuleren")
endmethod
```

   Beide methodes doen hetzelfde: een waarde en de programmabesturing teruggeven aan het aanroepend formulier. Deze methodes worden uitgebreider besproken in de paragraaf "Werking" in de volgende les.
7. Sluit het formulier en sla uw veranderingen op.

# Een dialoogvenster beheren

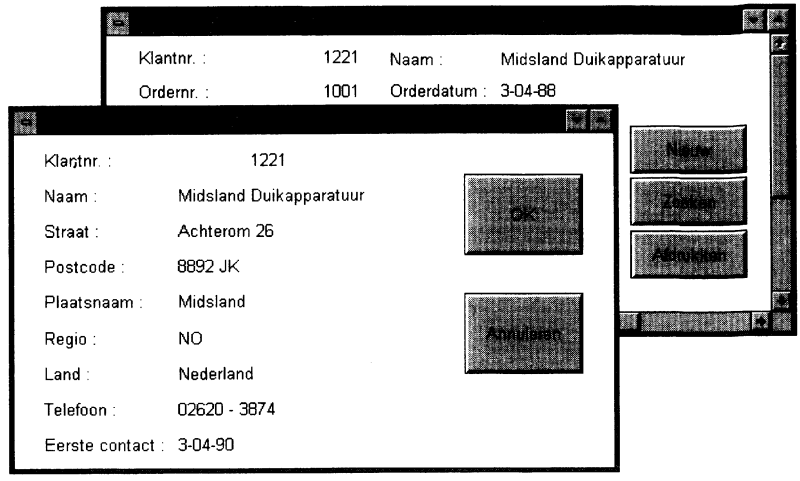
In deze les wordt beschreven hoe u een dialoogvenster beheert (of een ander formulier dat u met ObjectPAL wilt besturen). De volgende onderwerpen werden besproken:

- Het dialoogvenster openen en weergeven
- Een waarde uit het dialoogvenster halen
- Het dialoogvenster sluiten

In deze les wordt het formulier *Order* als aanroepend formulier gebruikt. Dit formulier roept het formulier *Klant* aan. De code die het dialoogvenster aanroept, is gekoppeld aan de knop met het label *Nieuw*. Als u op de knop *Nieuw* drukt, voegt deze code een nieuw, leeg record in, wordt er een uniek ordernummer gegenereerd en wordt het formulier *Klant* geopend. In deze les wordt het formulier *Klant* ook gebruikt om gegevens voor de klant in te voeren of op te vragen en om waarden terug te geven aan het aanroepend formulier.

In Afbeelding 6-1 ziet u de twee formulieren in werking.

Afbeelding 6-1 *Order* als aanroepend formulier




De eerste stap is de code te schrijven die het dialoogvenster aanroept. In deze les is de code gekoppeld aan de knop *Nieuw* in het formulier *Order*. In Voorbeeld 6-3 worden de benodigde stappen beschreven.

## Voorbeeld 6-3 Een dialoogvenster beheren

1. Open het formulier *Order* in het ontwerpvenster.
2. Inspecteer de knop met het label *Nieuw* en verander de naam in *nwKnop*.
3. Inspecteer *nwKnop* en kies 'Methodes' om het methodevenster te openen.
4. Dubbelklik op **pushButton** om een Editor-venster voor de ingebouwde methode **pushButton** te openen.
5. Bewerk de methode zodat deze er als volgt uitziet:

```
method pushButton(var eventInfo Event)
   var
      orderTbl Table
      nwKlantDlg Form
      dlgWaarde String
   endVar

   action(DataBeginEdit)
   action(DataInsertRecord)
   orderTbl.attach("ORDER.DB")
   Ordernr_.Value = orderTbl.cMax("Ordernr.") + 1
   action(DataPostRecord)
```

```
    if nwKlantDlg.open("klant") then
       dlgWaarde = nwKlantDlg.wait()
       if dlgWaarde = "OK" then
          Klantnr_.Value = nwKlantDlg.Klantnr_.Value
       endIf
       nwKlantDlg.close()
    else
        msgInfo("Probleem", "Kan dialoogvenster niet openen.")
    endIf
endmethod
```

6. Sla het formulier op en start het.
7. Klik op *nwKnop* om een nieuw record in te voegen, een nieuw ordernummer te genereren en het dialoogvenster te openen.
8. Gebruik het dialoogvenster om gegevens voor een nieuwe klant in te voeren of gegevens te vinden van een bestaande klant. Als u klaar bent, klikt u op de knop 'OK'.

***Opmerking*** U moet het toetsenbord gebruiken om door de records te schuiven. Dialoogvensters reageren niet op de menu's of de TurboBalken van Paradox.

9. Controleer, als het dialoogvenster is gesloten of *Klantnr.* een nieuwe waarde bevat.

## Werking

De code is onderverdeeld in drie blokken. Het eerste blok declareert variabelen, het tweede blok voegt een nieuw record in met een uniek ordernummer (met de methode die is beschreven in Voorbeeld 4-3) en het derde blok beheert het dialoogvenster.

*Regel 5* In het eerste blok luidt regel 5

```
nwKlantDlg Form
```

Deze regel declareert de variabele *nwKlantDlg* van het type Form. Net als de Table- en Report-variabelen, die zijn beschreven in de vorige lessen, verstrekt een Form-variabele een handle. Hier is *nwKlantDlg* een handle voor het dialoogformulier met de naam KLANT.FSL.

*Regel 15* Regel 15, de eerste regel in het derde codeblok, ziet er als volgt uit:

```
if nwKlantDlg.open("klant") then
```

Deze instructie probeert KLANT.FSL (of KLANT.FDL, als KLANT.FSL niet wordt gevonden) vanaf de schijf te laden en het formulier te starten. Als dit lukt, wordt de volgende coderegel uitgevoerd. Als het om de een of andere reden niet lukt, springt de uitvoering naar de **else**-clausule en verschijnt er een dialoogvenster dat u op de hoogte brengt van het probleem.

*Regel 16* Regel 16, de volgende regel in het derde blok, ziet er als volgt uit:

```
dlgWaarde = nwKlantDlg.wait()
```

Deze instructie zegt eigenlijk "Onderbreek de uitvoering van deze methode en wacht totdat *nwKlantDlg* een waarde teruggeeft. wijs die waarde vervolgens toe aan de variabele *dlgWaarde* en ga door met de uitvoering".

Een **wait**-instructie draagt de controle over aan een opgegeven formulier (in dit geval aan het dialoogvenster dat wordt vertegenwoordigd door *nwKlantDlg*). Terwijl het aanroepend formulier wacht op het aangeroepen formulier, reageert alleen het aangeroepen formulier op acties. Met andere woorden: het aangeroepen formulier is modaal.

***Belangrijk*** Het aanroepend formulier geeft de controle terug met een **formReturn**-instructie. In Voorbeeld 6-2 hebt u de volgende code gekoppeld aan de ingebouwde **pushButton**-methode van de knop 'OK' in het dialoogformulier:

```
method pushButton(var eventInfo Event)
   formReturn("OK")
endmethod
```

Deze code geeft de controle *en* de waarde "OK" terug aan het aanroepend formulier.

*Regel 17 tot en met 19* Regel 17 tot en met 19 in het derde blok zien er als volgt uit:

```
if dlgWaarde = "OK" then
      Klantnr_.Value = nwKlantDlg.Klantnr_.Value
endIf
```

Deze regels testen de waarde van *dlgWaarde*, de waarde die wordt teruggegeven door het dialoogvenster. Als *dlgWaarde* "OK" is, betekent dit dat de gebruiker op de knop 'OK' in het dialoogvenster heeft geklikt. De volgende instructie haalt de waarde van het veldobject *Klantnr.* op het formulier *nwKlantDlg* op en kent deze toe aan het kenmerk 'Value' van *Klantnr.*, een veldobject in het aanroepend formulier.

***Belangrijk*** In de volgende pseudocode ziet u hoe u een waarde haalt uit een object van een ander formulier.

```
objWaarde = formVar.objectNaam.Value
```

In dit voorbeeld is *objWaarde* een variabele waarin de waarde wordt opgeslagen, *formVar* is een Form-variabele (een handle voor het andere formulier), *objectNaam* vertegenwoordigt de naam van het object waarin u geïnteresseerd bent en Value specificeert het kenmerk 'Value'.

*Regel 20* Regel 20 luidt

```
nwKlantDlg.close()
```

Deze instructie sluit het dialoogvenster en verwijdert het van het scherm. Zonder een **close**-instructie zou u iedere keer dat deze

methode wordt uitgevoerd, een nieuwe kopie van het dialoogvenster openen.

## Samenvatting

In de lessen in dit hoofdstuk hebt u kennis gemaakt met de basismethoden voor het beheer van een multi-formulier applicatie. De volgende onderwerpen zijn behandeld:

- Een dialoogvenster maken door formulierkenmerken in te stellen
- Een formulier aanroepen vanuit een ander formulier
- Een **wait**-instructie gebruiken om de uitvoering in het aanroepende formulier te onderbreken en te wachten totdat het aangeroepen formulier een waarde teruggeeft
- Waarden halen uit het aangeroepen formulier
- Het aangeroepen formulier sluiten

HOOFDSTUK 7

# Buiten het gegevensmodel werken

Dit hoofdstuk is bedoeld voor programmeurs die kennis willen maken met een van de meer geavanceerde mogelijkheden van ObjectPAL. In dit hoofdstuk ziet u hoe u kunt werken met tabellen die niet zijn opgenomen in het gegevensmodel van een formulier, en zonder dat de tabellen worden weergegeven.

Het formulier *Order* dient als voorbeeld. Als u een bestelling plaatst voor een aantal produkten, moet u er zeker van zijn dat er genoeg van deze produkten in voorraad zijn. Als er genoeg zijn, moet u de hoeveelheid die u hebt besteld, aftrekken van de hoeveelheid in voorraad. U zou de tabel *Voorraad* kunnen toevoegen aan het gegevensmodel van het formulier, de juiste veldobjecten in het formulier kunnen plaatsen en hiermee rechtstreeks aan het werk gaan. Er kunnen zich echter situaties voordoen waarin u liever niet zo werkt, misschien omdat u het formulier eenvoudig en overzichtelijk wilt houden of omdat u wilt voorkomen dat een toevallige gebruiker toegang krijgt tot belangrijke informatie. In dergelijke situaties is het gebruik van een TCursor een uitstekend alternatief.

## Wat is een TCursor?

Een TCursor is een verwijzing naar de gegevens in een tabel. Met zo'n verwijzing kunt u gegevens op tabelniveau, op recordniveau en op veldniveau manipuleren, zonder dat de tabel hoeft te worden weergegeven. Als u een TCursor gebruikt, werkt u niet met een kloon of een kopie van de tabel Als u de records in een TCursor bewerkt, verandert de onderliggende tabel en eventuele vergrendelingen op de tabel hebben invloed op de TCursor.

***Belangrijk*** De TurboBalk heeft wel een hulpmiddel om een tabelframe te maken, maar niet een TCursor-hulpmiddel. Een TCursor is zuiver een

programmeerconstructie; een TCursor is in feite de belangrijkste constructie van ObjectPAL voor het werken met tabellen.

De relatie van een TCursor tot een tabel is te vergelijken met de relatie tussen een tekstcursor en een document van een tekstverwerkingsprogramma. In een tekstverwerkingsprogramma wijst de tekstcursor naar één letter tegelijk, kan de cursor zich door het hele document bewegen en geeft de cursor aan waar de bewerking plaatsvindt. Als u een TCursor opent naar een tabel, wijst de TCursor ook naar het huidige record, kan de TCursor zich naar elk record in de tabel verplaatsen en specificeert de TCursor welk record wordt bewerkt. Bovendien kunt u een TCursor gebruiken voor veel bewerkingen op tabelniveau.

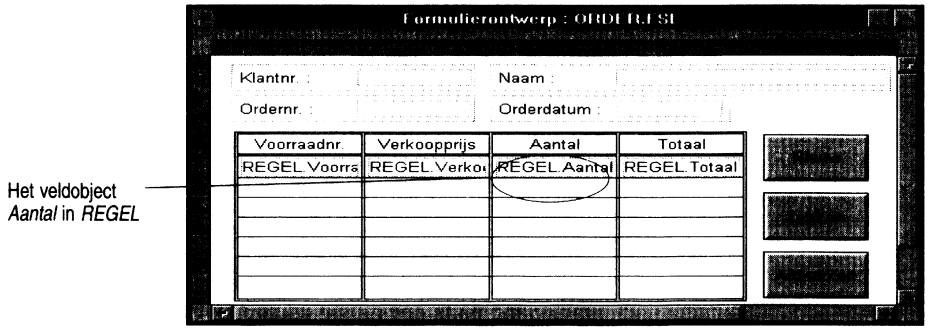
Als u een TCursor als variabele declareert en naar een tabel laat verwijzen, kunt u de TCursor gebruiken om de tabel te bewerken zonder dat de tabel wordt getoond. Het gebruik van een TCursor voor de bewerking van een tabel is te vergelijken met het gebruik van een afstandsbediening om op een TV een ander kanaal te kiezen. Als u een knop op de afstandsbediening indrukt, verandert de TV van kanaal. Als u een record in een TCursor bewerkt, verandert het record in de onderliggende tabel.

## Werken met een TCursor

In de volgende voorbeelden ziet u hoe u een TCursor gebruikt om de tabel *Voorraad* te onderhouden vanuit het formulier *Order*. U kunt het formulier *Order* gebruiken dat u in een vorige les hebt gemaakt. In deze les wordt dezelfde elementaire validiteitscontrole gebruikt als in Voorbeeld 5-3: code die is gekoppeld aan een ingebouwde **canDepart**-methode van een veldobject, controleert het kenmerk 'Value' van het object en houdt de cursor in het veld totdat de gebruiker een acceptabele waarde typt.

In Afbeelding 7-1 ziet u waar u de code kunt koppelen.

Afbeelding 7-1 Code koppelen aan het veldobject *Aantal* in *REGEL*




## Voorbeeld 7-1 Een TCursor gebruiken

1. Inspecteer in het tabelframe *REGEL* het veldobject met de naam *Aantal* en kies 'Methodes' om het methodevenster te openen.
2. Dubbelklik op **canDepart** om een Editor-venster te openen voor de ingebouwde **canDepart**-methode.
3. Bewerk de methode zodat deze er als volgt uitziet:

```
method canDepart(var eventInfo MoveEvent)
   var
      voorraadTC TCursor
      aantalAanwezig, aantalBesteld Number
   endVar

   voorraadTC.open("VOORRAAD.DB")
   voorraadTC.locate("Voorraadnr.", Voorraadnr_.Value)
   aantalAanwezig = voorraadTC.Aantal
   aantalBesteld = Self.Value

   if aantalBesteld < aantalAanwezig then
      aantalAanwezig = aantalAanwezig - aantalBesteld
      voorraadTC.edit()
      voorraadTC.Aantal = aantalAanwezig
      voorraadTC.endEdit()
   else
      msgInfo("Niet voldoende in voorraad",
              "Slechts " + String(aantalAanwezig) + " aanwezig.")
      eventInfo.setErrorCode(CanNotDepart)
   endIf
endmethod
```

4. Controleer de syntaxis en verbeter eventuele fouten.
5. Start het formulier en druk vervolgens op ***F9*** om de bewerkmodus te activeren.
6. Ga naar het veld *Aantal* in het tabelframe *REGEL* en typ een hoog getal voor de hoeveelheid (een getal groter dan 500 is voldoende). Het dialoogvenster wordt nu geopend en bevat de mededeling dat u een kleinere hoeveelheid moet invoeren.

## Werking

De code in deze methode is verdeeld in drie blokken. Het eerste blok declareert variabelen, het tweede blok opent een TCursor naar de tabel *Voorraad* en leest de waarde van het veld *Aantal* en het derde blok werkt de tabel *Voorraad* bij of toont een melding in een dialoogvenster, afhankelijk van het aantal dat in voorraad is.

*Regel 7* De zevende regel luidt

```
voorraadTC.open("VOORRAAD.DB")
```

Deze instructie opent de TCursor *voorraadTC* naar de tabel *Voorraad*. Nu kan deze methode *voorraadTC* gebruiken om te werken met gegevens in de tabel *Voorraad*. Als u een TCursor opent, wijst deze standaard naar het eerste record in de onderliggende tabel.

*Regel 8 en 9* De achtste en negende regel luiden

```
voorraadTC.locate("Voorraadnr.", Voorraadnr_.Value)
aantalAanwezig = voorraadTC.Aantal
```

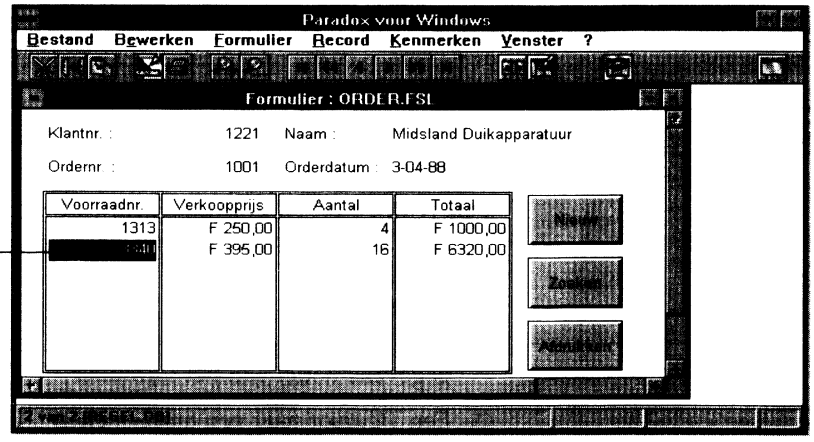
Regel 8 gebruikt de **locate**-methode om in het huidige record van het tabelframe *REGEL* te zoeken naar het voorraadnummer in de tabel *Voorraad*. De **locate**-methode die u gebruikt in een TCursor, is niet hetzelfde als de **locate** die u gebruikt om een tabelframe te doorzoeken, maar de methodes gedragen zich wel hetzelfde. In Voorbeeld 4-2 van Hoofdstuk 4 is al gezegd dat wanneer **locate** een tabelframe met succes doorzoekt, het formulier dat record activeert en het weergeeft in het tabelframe. Als **locate** de tabel van de TCursor met succes doorzoekt, wordt de TCursor eveneens verplaatst naar het record waar de waarde is gevonden.

Stel dat u tien eenheden van een artikel met het voorraadnummer 3340 bestelt, zoals in Afbeelding 7-2. In dit geval zou de **locate**-methode in de tabel *Voorraad* zoeken naar de waarde 3340 in het veld 'Voorraadnr.'

Als **locate** de waarde vindt, verwijst de TCursor naar dat record. (Als **locate** de waarde 3340 bijvoorbeeld vindt in record 30 van de tabel *Voorraad*, verwijst *voorraadTC* naar record 30.)

***Belangrijk*** Als de TCursor zich door de onderliggende tabel verplaatst, heeft dit *geen* invloed op het huidige record dat op het formulier wordt getoond. In Afbeelding 7-2 toont het formulier bijvoorbeeld record 2 van 2 in het tabelframe. Dit record blijft zichtbaar, ongeacht naar welk record de TCursor verwijst.

Afbeelding 7-2 Een record dat wordt getoond tijdens TCursor-**locate**




Omdat de **locate**-methode succesvol was en *voorraadTC* is verplaatst naar het juiste record, geeft regel 9 vervolgens de beschikbare hoeveelheid van het opgegeven voorraadnummer terug. Met andere woorden, voortbouwend op het vorige voorbeeld, als **locate** het voorraadnummer 3340 vindt in record 30, geeft regel 9 de waarde van het veld 'Aantal' voor record 30 terug en wordt deze waarde toegewezen aan de variabele *aantalAanwezig*.

*Regel 10* De tiende regel luidt

```
aantalBesteld = Self.Value
```

Strikt genomen is deze regel niet noodzakelijk. De regel is opgenomen om ervoor te zorgen dat de code die erop volgt gemakkelijker te lezen is. Deze regel haalt de waarde van het veldobject *Aantal* in het huidige record op het formulier en slaat deze op in de variabele *aantalBesteld*.

*Regel 12 en 13* Regel 12 en 13 vormen het begin van het derde codeblok.

```
if aantalBesteld < aantalAanwezig then
   aantalAanwezig = aantalAanwezig - aantalBesteld
```

Deze regels vergelijken de bestelde hoeveelheid met de hoeveelheid in voorraad. Als er genoeg in voorraad is, werken de volgende regels de tabel *Voorraad* bij. Is dit niet het geval, dan wordt naar de **else**-clausule gesprongen om een dialoogvenster te openen en te voorkomen dat het invoegpunt het veldobject verlaat.

*Regel 14 tot en met 16* Regel 14 tot en met 16 luiden

```
voorraadTC.edit()
voorraadTC.Aantal =aantalAanwezig
voorraadTC.endEdit()
```

Regel 14 activeert de bewerkmodus voor *voorraadTC* (en daarmee voor de tabel *Voorraad*). Regel 15 kent de bijgewerkte waarde van *aantalAanwezig* toe aan het veld 'Aantal' van het huidige record van *voorraadTC*. Regel 16 activeert voor *voorraadTC* de bewerkmodus.

## Samenvatting

Een TCursor is een krachtige instructie van ObjectPAL. Door middel van een TCursor kunt u met de gegevens in een tabel werken zonder de tabel weer te geven. In dit hoofdstuk zijn de volgende zaken aan de orde gekomen:

- Gegevens manipuleeren in een tabel die niet is opgenomen in het gegevensmodel van het formulier
- Een TCursor naar een tabel openen
- Een TCursor gebruiken om naar een waarde in een tabel te zoeken
- Een TCursor gebruiken om de onderliggende tabel te bewerken

DEEL II

# Grondbeginselen

In dit deel van deze handleiding wordt de Integrated Development Environment (IDE) besproken en wordt uitgelegd hoe u met ObjectPAL werkt.

Deel II bevat de volgende hoofdstukken:

- In Hoofdstuk 8, "ObjectPAL: overzicht", wordt een overzicht gegeven van objecten, acties, methodes en objectgeoriënteerd programmeren.
- In Hoofdstuk 9, "De ObjectPAL-Editor", wordt uitgelegd hoe u de ingebouwde Paradox-Editor gebruikt om ObjectPAL-code te schrijven en te wijzigen.
- In Hoofdstuk 10, "De ObjectPAL-Debugger", wordt uitgelegd hoe u de ingebouwde debugger gebruikt om uw code te testen.
- In Hoofdstuk 11, "Taalstructuur", wordt de algemene structuur en syntaxis van ObjectPAL-code besproken, met inbegrip van benoemingsconventies voor en het gebruik van variabelen en constanten.

HOOFDSTUK 8

# ObjectPAL: overzicht

Dit hoofdstuk geeft een overzicht van de ideeën die aan de taal ObjectPAL ten grondslag liggen en de voordelen van deze taal. Daarnaast wordt er een strategie voor objectgeoriënteerd programmeren geboden.

## Programmeren in ObjectPAL

Programmeren in ObjectPAL lijkt in bepaalde opzichten op programmeren in andere talen, maar wijkt er ook op een aantal manieren van af. ObjectPAL lijkt op traditionele programmeertalen, omdat de taal variabelen gebruikt, controlestructuren kent, zoals **if...then...else**, **for**-lussen, en **while**-lussen, berekeningen uitvoert en de mogelijkheid biedt om functies te maken (in ObjectPAL worden functies *methodes* en *procedures* genoemd).

ObjectPAL wijkt van traditionele programmeertalen af, omdat het op objecten is gebaseerd. Bij een traditionele programmeertaal is programmeren een zaak van alles of niets. Of u hebt van het begin tot het einde controle over de applicatie of u programmeert helemaal niets. Met ObjectPAL staat u echter niet voor zo'n enorme taak. Omdat ObjectPAL zich op objecten richt, kunt u zo veel of zo weinig objecten programmeren als u wilt.

De objecten waarvoor u ObjectPAL-code schrijft, zijn de objecten waarmee u al werkte. Wilt u dat Paradox de waarde controleert die in een veld wordt ingevoerd en dat er een pieptoon klinkt als deze waarde niet klopt? U kunt deze functie eenvoudig programmeren door de ingebouwde code te veranderen die wordt uitgevoerd zodra de waarde van het veld verandert. Deze handeling is snel te leren en is ook in andere situaties gemakkelijk toepasbaar. Natuurlijk is niet alles wat u met ObjectPAL kunt doen zo eenvoudig. U bepaalt echter zelf hoeveel of hoe weinig u wilt programmeren.

# Wat is ObjectPAL?

Formeel gesproken is ObjectPAL een geavanceerde actiegestuurde objectgeoriënteerde visuele programmeertaal. U kunt ObjectPAL gebruiken om een volledig eigen applicatie te maken met totaal nieuwe knoppen, menu's, dialoogvensters, prompts, waarschuwingen en helpfuncties. U kunt een gebruikersinterface maken voor een database-applicatie of ObjectPAL gebruiken om een applicatie te maken die niets meer met databases te maken heeft.

## Een uitbreiding van Paradox

Los van de formele definities en de ambitieuze doelstellingen kunt u ObjectPAL het best zien als een hulpmiddel dat een aanvulling vormt op de interactieve mogelijkheden van Paradox. Als u ObjectPAL als een uitbreiding van Paradox beschouwt, is het niet moeilijk manieren te bedenken om met ObjectPAL taken uit te voeren die zonder deze programmeertaal vervelend, moeilijk, tijdrovend of zelfs onuitvoerbaar zouden zijn.

*Terugkerende taken automatiseren*

Stel dat u unieke, maar opeenvolgende identificatienummers wilt maken als gebruikers in een netwerkomgeving een nieuw factuur-record openen. Als de laatstgemaakte factuur bijvoorbeeld het nummer 1203 heeft, moet het nummer van de volgende factuur dus 1204 zijn. Zonder ObjectPAL zou u eerst een één-record en een één-veld tabel kunnen maken in een gemeenschappelijke gegevensdirectory en het laatst gebruikte identificatienummer in dat veld kunnen opslaan. De gebruikers van het netwerk moeten dan die tabel openen, de bewerking starten, het record vergrendelen, het identificatienummer veranderen in het eerstvolgende nummer, het record weer ontgrendelen, de bewerkmodus verlaten, de tabel sluiten, terugkeren naar de factuurtabel en het nieuwe identificatienummer invoeren. Met ObjectPAL kunt u echter een applicatie maken die deze stappen automatisch uitvoert zodra een gebruiker een nieuwe factuur maakt en deze doorvoert.

*Veldopmaak corrigeren*

Gedetailleerde wijzigingen uitvoeren in velden kan moeilijk zijn als u queries interactief gebruikt. Als een telefoonnummerveld bijvoorbeeld in een tabel is ingevoerd of geïmporteerd met een streepje dat het netnummer scheidt van de rest van het nummer, kunt u de opmaak onmogelijk met een query corrigeren. De enige mogelijkheid zou dan zijn de records een voor een aan te passen. Met ObjectPAL kunt u echter een routine schrijven die het telefoonveld van elk record bekijkt en alle velden corrigeert die niet goed zijn ingevoerd.

*Gegevens beveiligen*

Soms zult u gebruikers willen waarschuwen dat zij op het punt staan iets te doen dat de database kan beschadigen (zoals een sleutelveld veranderen). De structuur van uw database (hoe u de tabellen hebt gekoppeld, hoe u de referentiële integriteit hebt geregeld en het type

veldvalidatie dat u hebt gedefinieerd) zorgt vaak voor een redelijke bescherming van de validiteit van de gegevens. Soms is er echter een gespecialiseerdere vorm van beveiliging nodig en die is niet te bereiken zonder ObjectPAL.

*De kracht van Paradox*

Houd er echter rekening mee dat u veel handelingen die u met een database kunt verrichten, met Paradox interactief kunt uitvoeren. Als u ObjectPAL alleen gebruikt om een lastig probleem in de verwerking van uw gegevens op te lossen, kunt u beter eerst bekijken of u dit probleem niet interactief met Paradox kunt oplossen. Zelfs ervaren gebruikers van de interactieve kant van Paradox hebben nog maar een deel van de kracht van queries en berekeningen ontdekt. Bovendien biedt het programma interactieve mogelijkheden voor de validiteitscontrole van gegevens, voor opzoekdefinities, keuzelijsten en veel andere krachtige functies van de gebruikersinterface.

## Objectgeoriënteerd

ObjectPAL werkt met objecten. U maakt en gebruikt objecten, zoals velden, lijnen, ellipsen, vakken en tabelframes, als u formulieren en rapporten ontwerpt. Volgens de formele definitie bestaat een object uit gegevens en code. In ObjectPAL-termen hebben objecten kenmerken (zoals kleur, positie en lijndikte) en methodes (code die bepaalt hoe het object zich gedraagt). Kenmerken zijn gegevens en methodes zijn code.

*Kenmerken*

Het belangrijkste dat u moet onthouden, is dat *objecten kenmerken hebben*. Als u een object maakt, krijgt het kenmerken die de weergave en het gedrag van het object definiëren. Zo heeft een vak bijvoorbeeld de kenmerken formaat, positie, kleur en frame. Als u ObjectPAL gebruikt, kunt u alle kenmerken instellen of wijzigen die u ook interactief in Paradox kunt gebruiken. U kunt bijvoorbeeld op een interactieve manier een groot blauw vak maken en dit met ObjectPAL veranderen in een klein rood vak.

*Context*

U moet er ook rekening mee houden dat *objecten zich in een context bevinden*.

De context van een bepaald object wordt gedefinieerd door de objecten waarin het zich bevindt. Deze eigenschap van ObjectPAL biedt ervaren programmeurs veel flexibiliteit en krachtige mogelijkheden. Als beginnend ObjectPAL-programmeur hoeft u alleen te weten dat het formulier alle andere objecten insluit. Als u objecten op een formulier plaatst, geeft u deze objecten een context.

*Visuele programmering*

Deze methode om objecten te plaatsen wordt *visuele programmering* genoemd, omdat het formulier u tijdens de programmering de gebruikersinterface van uw applicatie laat *zien*. Als u een ObjectPAL-applicatie wilt maken, plaatst u objecten, zoals velden, keuzelijsten, afrollijsten, knoppen en pictogrammen, op een formulier en stelt u de kenmerken van deze objecten in. Als de opmaak van het

formulier u bevalt, gebruikt u ObjectPAL om alleen het gedrag te veranderen van de objecten waarvan het standaardgedrag niet overeenkomt met uw bedoeling.

Dit werk verloopt heel anders dan wanneer u een applicatie zou maken in een niet-visuele omgeving. In veel andere programmeertalen maakt u de gebruikersinterface door code te schrijven in een tekst-editor. Als u de applicatie wilt starten, moet u de code compileren, eventuele fouten opsporen en de applicatie opnieuw starten. U moet deze procedure herhalen totdat de applicatie werkt. Telkens als u de positie van een element wilt veranderen, hebt u toegang nodig tot de code. Hierdoor kunnen nog meer fouten ontstaan.

*Objectenschema's*

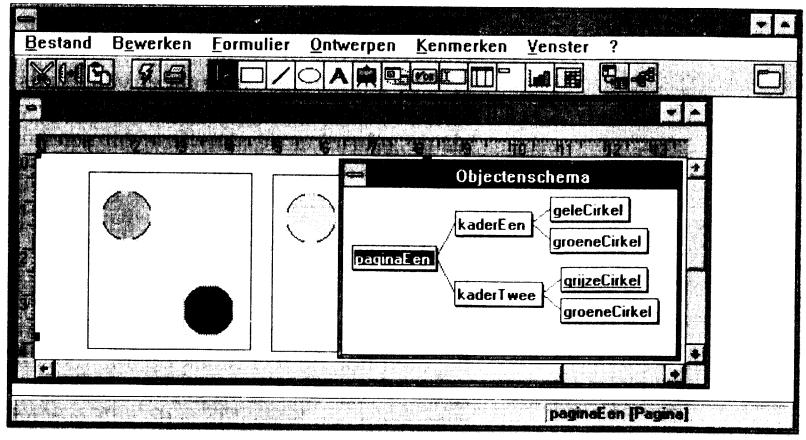
U kunt de relatie tussen de objecten niet alleen op het formulier bekijken, maar ook in een Objectenschema. Objectenschema's geven een conceptueel overzicht van de relatie tussen objecten.

Via Objectenschema's hebt u ook toegang tot de code van een object. Als een Objectenschema van een formulier is geopend, ziet u alle objecten die zich op het formulier bevinden en alle objecten binnen deze objecten. Als u een Objectenschema van een formulier wilt weergeven, gaat u als volgt te werk:

1. Selecteer de pagina door ergens op de pagina te klikken (niet op een object).
2. Druk op *Esc* om het formulier te selecteren.
3. Open een Objectenschema door te klikken op de knop 'Objectenschema' (of kies 'Formulier | Objectenschema').

Als u een object in het Objectenschema wilt inspecteren, klikt u met de rechtermuisknop op de naam van het object in een Objectenschema. Een onderstreepte objectnaam geeft aan dat er eigen code aan het object is gekoppeld. Stel dat een formulier een pagina insluit, dat die pagina twee vakken insluit en dat elk vak weer twee cirkels insluit. Afbeelding 8-1 toont het formulier met het bijbehorende Objectenschema.

Afbeelding 8-1 Gebruik van het Objectenschema




## Actiegestuurd

In Windows heeft geen enkel programma absolute, permanente controle. Geen enkel programma kan ervan uitgaan dat het uit de eerste hand informatie heeft over de specifieke hardware van een systeem of dat het informatie heeft over andere programma's en wat die doen. Elke applicatie kan op praktisch elk moment worden gestopt als de gebruiker op een andere applicatie klikt.

Deze interface is om twee redenen mogelijk: op de eerste plaats is elke applicatie volledig van Windows afhankelijk voor verwerkingstijd, schermruimte en andere bronnen. Uiteindelijk beheert Windows alle Windows-applicaties. Op de tweede plaats nodigt de aard van Windows uit tot *actiegestuurde* Windows-applicaties.

*De actiegestuurde interface*

Een actiegestuurde interface reageert alleen op specifieke handelingen van het systeem of van de gebruiker, zoals muis-bewegingen. De applicatie neemt de besturing van het systeem (via Windows) lang genoeg over om op een actie te kunnen reageren. Vervolgens wacht de applicatie op de volgende actie. Als u deze gedachte doortrekt, kunt u elk object als een kleine applicatie zien.

Bij een uitgebreidere beschrijving van objecten komt hun actiegestuurde karakter aan de orde. Objecten hebben een context die hun relatie tot andere objecten bepaalt, een verzameling kenmerken die hun eigenschappen bepaalt en ingebouwde methodes die hun reactie op acties bepalen.

*Ingebouwd gedrag*

Als u een rechthoek tekent in een formulier, tekent u niet alleen, maar zet u ook de eerste programmeerstap. De rechthoek is een object met kenmerken, dat bestaat in de context van een formulier. De rechthoek

heeft echter ook een gedrag. De rechthoek is namelijk zo gemaakt dat deze op acties reageert.

De rechthoek reageert standaard niet op de meeste acties. U gebruikt ObjectPAL om de rechthoek te vertellen dat er iets anders of iets meer dan de standaardreactie moet volgen als er een bepaalde actie optreedt. U kunt bijvoorbeeld de kleur van de rechthoek veranderen als de gebruiker de aanwijzer binnen het kader van de rechthoek plaatst en de kleur weer terug veranderen als de aanwijzer de rechthoek verlaat.

Als ObjectPAL-programmeur definieert u de reacties van objecten opnieuw. Met andere woorden: u hoeft er niet voor te zorgen *dat* de rechthoek reageert. Reageren is al eigen aan de rechthoek. U geeft alleen aan *hoe* de rechthoek moet reageren. Verder hoeft u ook niet te bepalen hoe de rechthoek op alle mogelijke acties moet reageren. U geeft alleen aan hoe de rechthoek moet reageren als u de standaard-reactie op een bepaalde actie wilt veranderen.

*Ingebouwde methodes en standaardreacties*

Elk object beschikt over een verzameling standaardreacties. Als u de standaardreactie van een object wilt wijzigen, verandert u een of meer *ingebouwde methodes* van het object. In ObjectPAL bent u het grootste deel van de tijd bezig met het wijzigen van ingebouwde methodes.

*Types*

ObjectPAL groepeert alle objecten die u met behulp van de TurboBalk tekent in een categorie met de naam UIObjecten. (In ObjectPAL worden objectcategorieën *types* genoemd. Sommige andere programmeertalen noemen deze categorieën klassen.) Een pagina van een formulier is, net als het formulier zelf, een UIObject. Formulieren vertonen gedrag dat verder gaat dan hun gedrag als UIObject. Voorlopig kunt u formulieren echter als UIObjecten beschouwen. De term UIObject staat voor User Interface Object, oftewel een object in de gebruikersinterface.

UIObjecten hebben hun eigen verzameling ingebouwde methodes. Deze ingebouwde methodes worden automatisch geactiveerd in antwoord op acties of handelingen. Als u bijvoorbeeld binnen de grenzen van een object klikt, roept Paradox de ingebouwde **mouseClick**-methode van dat object aan.

De methodes die in een object zijn ingebouwd, zijn afhankelijk van het soort object. Afgezien van een paar opmerkelijke uitzonderingen, hebben de meeste UIObjecten een zelfde basisverzameling. Zo hebben veldobjecten een ingebouwde **changeValue**-methode, terwijl kaders of vensters deze niet hebben. De **changeValue**-methode wordt uitgevoerd zodra de waarde in een veldobject verandert. Een **changeValue**-methode heeft voor een kader geen zin, omdat een kader geen waarden kan bevatten.

Paradox maakt het zoeken en wijzigen van ingebouwde methodes van een object gemakkelijk. U inspecteert het object door het menu 'Kenmerken' te openen. Daarna kiest u de ingebouwde methode die u wilt wijzigen. Er wordt een venster van de ObjectPAL-Editor geopend en de cursor bevindt zich op de positie waar u kunt beginnen te typen.

## Modulair

*Objecten zijn onafhankelijk*

In het begin van dit hoofdstuk werd uitgelegd dat u met ObjectPAL zelf kunt bepalen wat u wel en niet wilt programmeren. ObjectPAL beschikt over deze flexibiliteit omdat objecten inherent modulair zijn. Met andere woorden: *objecten zijn onafhankelijk*. U kunt het gedrag van een object op een formulier veranderen zonder het gedrag van de overige objecten op het formulier te veranderen.

Deze eigenschap heeft meer gevolgen dan u misschien denkt. Een evident gevolg is dat objecten gemakkelijk te programmeren zijn. Een niet voor de hand liggend gevolg is dat u ObjectPAL kunt gebruiken om ingewikkelde systemen te bouwen. In een niet-object-georiënteerde programmeertaal is het een algemene regel dat hoe groter of ingewikkelder een systeem wordt, hoe groter de kans is dat het systeem onstabiel wordt, en niet alleen omdat het systeem groter is, meer coderegels telt en daardoor meer programmeerfouten bevat.

In traditionele programmeertalen werden systemen zo ontwikkeld dat alle intelligentie zich boven of bijna boven in het systeem bevindt. Processen in een systeem worden als lineair gezien en de programmering wordt op een lineaire manier uitgevoerd. Ingewikkelde systemen uit de werkelijkheid, zoals verkeerspatronen en de aandelenmarkt, werken volgens organische wetten, bijvoorbeeld: niets beweegt in een rechte lijn en er komt weinig verandering vanuit de top van het systeem. In de werkelijkheid wordt de besturing niet alleen van boven naar beneden uitgevoerd, maar ook vanaf de basis, vanuit de interactie tussen alle kleine eenheden, de zogenaamde *subsystemen*.

*Realistisch gedrag*

Objectgeoriënteerde programmering geeft u de mogelijkheid systemen te ontwikkelen vanuit een meer organische benadering. In een objectgeoriënteerd systeem bouwt u de intelligentie in de kleine eenheden (de objecten). Als u aandacht besteedt aan correct gedrag van de subsystemen, zal het hele systeem intuïtiever overkomen en veel meer een afspiegeling zijn van de werkelijkheid.

*Onafhankelijkheid vermindert de kans op fouten*

Daarnaast ondersteunt een objectgeoriënteerd systeem een proces van stapsgewijze ontwikkeling, een proces waarbij u telkens naar het programma kunt terugkeren om het te verfijnen. U kunt een object intelligenter maken zonder het hele systeem in gevaar te brengen. U kunt achteraf de code van een object wijzigen, omdat het object betrekkelijk onafhankelijk is. Voor de doorgewinterde programmeur is het voordeel van deze mogelijkheden duidelijk: voor het onderhoud en de verbetering van een objectgeoriënteerd systeem

hoeft de programmeur niet meer het hele systeem te kennen. In een goed ontworpen systeem is een kleine verandering ook werkelijk een kleine verandering.

U hoeft geen objectgeoriënteerde ontwerpprincipes te kennen om te kunnen beginnen met programmeren in ObjectPAL. Als u met kleine stukjes begint en deze vervolgens verder uitwerkt, wordt het ontwerp van uw systeem beter dan wanneer u alles van bovenaf probeert te beheersen. Het mooie van ObjectPAL is dat de beste manier om de taal te gebruiken ook de gemakkelijkste is:

- Plaats objecten op een formulier.
- Stel de kenmerken van deze objecten in.
- Koppel indien nodig eigen code aan enkele ingebouwde methodes van deze objecten.

U kunt programma's op de ouderwetse manier schrijven door bovenaan te beginnen en omlaag te werken. U maakt dan echter geen gebruik van de ware kracht van ObjectPAL. Als u uw applicatie daarentegen modulair houdt, zodat de code die betrekking heeft op een object zich zo dicht mogelijk bij dit object bevindt, krijgt u een goed ontworpen en gemakkelijk te onderhouden applicatie.

# ObjectPAL voor programmeurs

In de volgende paragrafen wordt een beschrijving gegeven van ObjectPal voor ervaren programmeurs. Een aantal termen zal u misschien onbekend zijn. Deze termen worden verderop in dit hoofdstuk of in de volgende hoofdstukken in meer detail uitgelegd.

Hoewel ObjectPAL toegankelijk is voor onervaren programmeurs, is het voor serieuze ontwikkelaars belangrijk te weten dat de taal zeer uitgebreid en volledig ontwikkeld is. De taal is hierdoor ook geschikt voor veeleisende programmeurs die geavanceerde applicaties willen schrijven.

## Kenmerken van de taal

ObjectPAL ondersteunt de volgende functies:

- Ingebouwde afhandeling van acties
- Sterke type-indeling van gegevens
- Door de gebruiker gedefinieerde gegevenstypes
- Krachtige gegevenstypes, zoals arrays met te wijzigen grootte, associatieve arrays (*dynamische arrays* genoemd) en records
- Gestructureerde programmabesturing

- Een uitgebreide bibliotheek met methodes en procedures
- Door de gebruiker gedefinieerde methodes en procedures
- Aanroepen van functies en procedures die zijn geschreven in andere programmeertalen, zoals Pascal, C en C++
- Aanroepen van een extern gecompileerd helpsysteem (gecompileerd met een Windows help-compiler)

## Besturingsfuncties

Met ObjectPAL kunt u zowel toetsaanslagen als muishandelingen onderscheppen en veranderen. Meestal hoeft u de besturing op een dergelijk laag niveau niet in te bouwen. In plaats hiervan kunt u code schrijven die op *acties* reageert. Een actie is een bericht aan een object, dat door een bepaalde handeling wordt gegenereerd (bijvoorbeeld door een toetsaanslag of een muisklik).

*Acties beheren*

U kunt alle muisacties opvangen, beantwoorden, veranderen, maken en simuleren, zoals de positie van de muis, klikken met de linker of de rechter muisknop, dubbelklikken en klikken in combinatie met *Shift*, *Alt* of *Ctrl*. Over toetsaanslagen hebt u een soortgelijke controle.

U kunt formulieren en alle andere weergave-objecten openen, verplaatsen, van formaat veranderen, tot een pictogram verkleinen, het maximale formaat geven en op andere manieren manipuleren. U kunt meervoudige formulieren als dialoogvensters gebruiken of als modules voor een applicatie.

*Kenmerken instellen*

Elk objectkenmerk dat u interactief kunt instellen (bijvoorbeeld kleur), kunt u ook in ObjectPAL instellen. Als een formulier actief is, kan ObjectPAL veel kenmerken beheren die niet beschikbaar zijn in het menu 'Kenmerken', zoals de positie van een object en de focus. U kunt bijvoorbeeld een gebruiker op een formulier "volgen" door een gekleurd kader te tekenen om het object dat de focus heeft.

U kunt menu's op het hoogste niveau maken met bijbehorende afrolmenu's. U kunt de TurboBalk verbergen en zelfs de hoofdtitel van het bureaublad van Paradox veranderen.

*Tabellen manipuleren*

Elke tabelhandeling die normaal interactief beschikbaar is, is ook beschikbaar in ObjectPAL. Daarnaast zijn er nog talrijke handelingen in ObjectPAL beschikbaar waarover u interactief niet kunt beschikken. U kunt bijvoorbeeld tabellen manipuleren als Tables, TableViews, TableFrames en TCursors. Een *Table* gebruikt u voor functies als optellen en aftrekken. Een *TableView* is een tabel die is geopend in een venster (dat Paradox maakt als u 'Bestand | Openen | Tabel' kiest). Een *TableFrame* is een tabelobject dat op een formulier is geplaatst. Een *TCursor* is een tabel-handle die u achter de schermen kunt gebruiken. TCursors zijn handig bij zoeken en sorteren.

*Het bestandssysteem beheren*

Alle functies van het bestandssysteem die interactief beschikbaar zijn, zijn ook beschikbaar in ObjectPAL. U kunt de ingebouwde Blader-modus aanroepen, zodat de gebruiker een bestand kan kiezen. U kunt bestanden verwijderen of hernoemen en directories aanmaken.

*Queries uitvoeren op gegevens*

U kunt interactief queries maken(.QBE-bestanden) en deze queries vervolgens uitvoeren in ObjectPAL. U kunt query-opdrachten ook direct in ObjectPAL schrijven. Deze queries kunnen variabelen bevatten die tijdens de uitvoering worden geëvalueerd.

# Objectgeoriënteerd programmeren

De objectenterminologie kan voor de ervaren programmeur verwarrender zijn dan voor de beginnende programmeur. Hoe beter u een programmeertaal of een paradigma kent, des te moeilijker het is om er nieuwe termen voor te leren gebruiken. Voor objectgeoriënteerd programmeren hoeft u echter niet veel te leren.

In het dagelijkse taalgebruik zijn objecten niet meer dan dingen. Voor een programmeur is een object samengesteld uit nauw verbonden gegevens en code. U kunt objecten zowel interactief als in ObjectPAL maken.

Als u objectgeoriënteerde programmering goed wilt laten verlopen, moet u beginnen met de objecten. U kunt het best beginnen met de objecten die u al kent, namelijk de objecten die u op een formulier plaatst.

Probeer geen ingewikkelde multi-formulier applicaties te maken met toetsenbordroutines op laag niveau, vervolgmenu's en massa's dialoogvensters. U leert ObjectPal beter kennen als u niet te groot denkt, althans niet in het begin. Voor uw eerste project kunt u beter beginnen met het maken van specifieke objecten op specifieke formulieren, die taken uitvoeren die u nodig hebt. Al doende ontdekt u vanzelf de mogelijkheden die u voor uw *volgende* project kunt benutten.

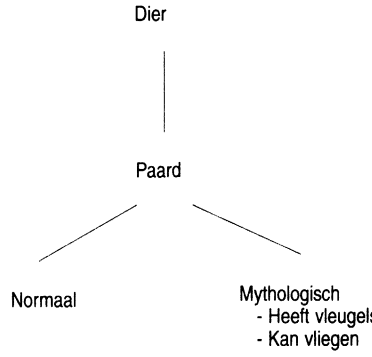
## De taal van objecten

Zonder dat u het zich misschien realiseert, weet u al veel over objecten en objecttypes. Iedereen denkt en spreekt al in termen van types en dingen, bijvoorbeeld: dier, plant of mineraal. Types kennen *hiërarchieën*. Een paard is bijvoorbeeld een type dier. Objecten worden onderverdeeld naar *type*. Pegasus is bijvoorbeeld een soort paard. Pegasus is dus een object van het type paard van het type dier.

In normale taal is Pegasus een mythologisch paard dat kan vliegen. In objectgeoriënteerde termen is Pegasus een object van het type mythologisch paard (van het type paard van het type dier) dat alle *kenmerken* en functies van een paard erft en waaraan enkele unieke

eigenschappen zijn toegevoegd, zoals de eigenschap vliegen (zie Afbeelding 8-2). Dank zij het objectgeoriënteerde paradigma kunt u zaken in een programma op ongeveer dezelfde manier indelen als in gewone taal.

Afbeelding 8-2 Een hiërarchie van types en objecten




In objectgeoriënteerde termen zou u, als de ontwerper van het type paard het type correct heeft geschreven, het object Pegasus via de volgende instructie opdracht kunnen geven om te springen, zonder dat u weet hoe het komt dat het paard kan springen:

```
Pegasus.spring()
```

U hoeft alleen te weten dat Pegasus een paard is en dat u wilt dat het paard springt.

*De procedurele benadering*

Bij procedurele programmering bepalen functies de werking van het programma. Stel dat u met twee types dieren werkt: paarden en dolfijnen. De procedure om een paard te laten springen is dan heel anders dan de procedure om een dolfijn te laten springen. In een procedurele taal, zou de functie **spring** maar één keer worden gecodeerd, bijvoorbeeld als volgt:

```
spring(hetDier)
   if hetDier = "Paard" then
      loopRichting = voorwaarts
      voorBenen(omhoog)
      achterBenen(duw)
   else if hetDier = "Dolfijn" then
      zwemRichting = omhoog
      zwemSnelheid = snel
      staart(sla)
      else
         message("Verkeerde gegevens ingevoerd voor deze functie.")
      endif
   endif
```

In dit voorbeeld moet één functie een groot aantal verschillende types gegevens verwerken. Beslissingen worden binnen de functie genomen. De effectiviteit van de functie is grotendeels afhankelijk van de vooruitziende blik van de programmeur, namelijk van zijn vermogen te voorzien hoeveel verschillende gegevens de functie zal moeten verwerken. Als programma's verder worden ontwikkeld, zal de functie **spring** worden gebruikt voor meer diersoorten die de programmeur misschien niet allemaal heeft voorzien. In een procedurele taal schrijft u één functie die een aantal verschillende dingen moet doen.

*De objectgeoriënteerde benadering*

In een objectgeoriënteerde taal laat u de taalverwerker beslissen welke *methode* wordt gebruikt, afhankelijk van het type object dat de methode aanroept. De onderliggende programmacode wordt speciaal voor objecten geschreven. In objectgeoriënteerde programmeertermen zou de algemene methode **spring** worden genoemd, maar zou er één methode zijn voor paarden en een andere voor dolfijnen. U moet dus aangeven welk dier moet springen. Deze instructie laat bijvoorbeeld een paard springen:

```
paard.spring()
```

Deze instructie laat een dolfijn springen:

```
dolfijn.spring()
```

U hoeft alleen te onthouden met welk dier u werkt (paard of dolfijn) en wat het dier moet doen (springen). De taal zorgt voor de rest.

## Objecten en ObjectPAL

Net als de dieren in deze voorbeelden worden ObjectPAL-objecten ingedeeld in types. Objecten van een bepaald type hebben dezelfde kenmerken en methodes. Alle tekstbestanden hebben gemeenschappelijke kenmerken en ook alle tabellen hebben gemeenschappelijke kenmerken, maar de kenmerken van tabellen en tekstbestanden zijn verschillend. Tekstbestanden en tabellen zijn dus objecten van verschillende types. Daarom gebruikt u een aparte groep methodes voor de bewerking van tekstbestanden en een andere groep voor de bewerking van tabellen.

Methodes voor verschillende objecttypes kunnen dezelfde naam hebben. Net als in het vorige voorbeeld, waarin **spring** voor zowel paarden als dolfijnen werd gebruikt, gebruikt ObjectPAL één **open**-methode om een tabel te openen en een andere om een tekstbestand te openen. De syntaxis van ObjectPAL lijkt op die van de menselijke taal. Stel dat u denkt "Ik wil een tabel openen" of "Ik wil een tekstbestand openen". De onderliggende code voor het openen van een tabel is anders dan de code voor het openen van een tekstbestand, maar het werkwoord "openen" blijft hetzelfde en meer hoeft u niet te onthouden. ObjectPAL start de **open**-methode die overeenkomt met het objecttype. U hoeft dus niet, zoals bij een

procedurele taal, twee verschillende opdrachten te onthouden (bijvoorbeeld OpenTable en OpenTextFile).

Ook hoeft u code die op een object werkt, niet te koppelen aan dat object. De code voor het openen van een tabel kan bijvoorbeeld worden gekoppeld aan een object, een knop, een veldobject of een ander object dat u op een formulier kunt plaatsen. Als de code wordt uitgevoerd, wordt de tabel geopend. Waaraan u de code koppelt, hangt af van wanneer u de code wilt uitvoeren.

Het is ook belangrijk te weten dat niet elke methode geschikt is voor elk objecttype. Het is bijvoorbeeld onzin om een slak te laten springen met **slak.spring()**.

Hetzelfde geldt voor Paradox en ObjectPAL: objecten van verschillende types reageren anders op acties en niet elke methode is geschikt voor elk objecttype. De resultaten van een ObjectPAL-instructie, zoals de volgende, variëren afhankelijk van het type van *ditObject*.

```
ditObject.open("order")
```

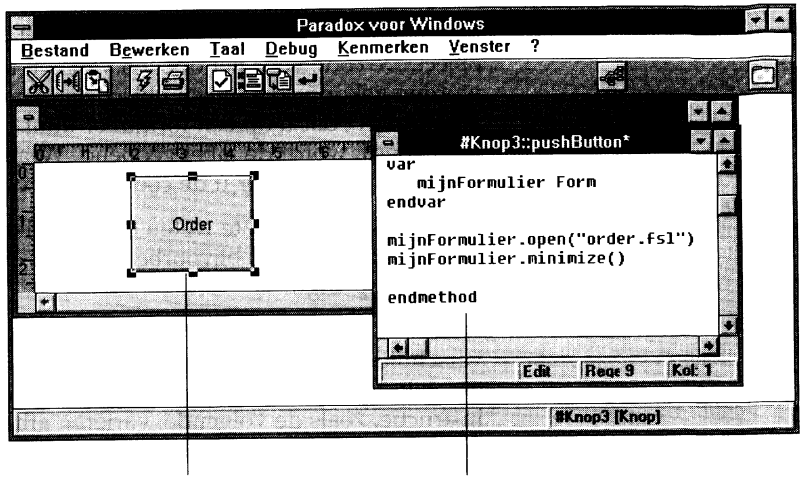
Als u applicaties ontwikkelt, gebruikt u een aparte groep methodes voor de bewerking van tabellen en een andere groep methodes voor de bewerking van tekstbestanden, omdat tabellen en tekstbestanden objecten zijn van verschillende types.

Als u ObjectPAL gebruikt, moet u zich afvragen met welk *object* u werkt en wat u met het object wilt *doen*. Als u een antwoord weet op deze vragen, weet u welk objecttype en welke methodes u moet gebruiken.

## Objectgeoriënteerde strategie

Het maken van Paradox-applicaties is vooral een kwestie van *objecten* op formulieren plaatsen en ObjectPAL-*methodes* schrijven om te bepalen hoe deze objecten zich gedragen. Dit komt overeen met de kreet "Hé jij, doe dit". (Als u dit onsympathiek vindt klinken, kunt u ook denken in termen van objecten en handelingen of van zelfstandige naamwoorden en werkwoorden.) U geeft eerst een object op en daarna een bewerking (zie Afbeelding 8-3).

Afbeelding 8-3 Objecten en handelingen opgeven (Hé jij, doe dit)




Als u op deze knop klikt, geeft u de knop opdracht om de ingebouwde **pushButton**-methode uit te voeren

Binnen de methode geeft deze coderegel een object op (*mijnFormulier*) en een handeling (**minimize**)

In een traditioneel, procedureel programma dat niet interactief is, bepaalt de programmacode wat er gebeurt en wanneer. De uitvoering van een programma verloopt meestal lineair. Een programma kan bijvoorbeeld een tabel weergeven, een berekening maken, vervolgens de resultaten weergeven enzovoort, en dat alles onder de besturing van het programma.

ObjectPAL werkt anders. Een ObjectPAL-applicatie geeft de gebruiker de controle: gebruikers communiceren met objecten, zoals knoppen, tabellen, menu's enzovoort, in de volgorde die zíj kiezen. De procedurele benadering is niet langer van toepassing: u zult een objectgeoriënteerde strategie moeten toepassen.

*Plannen is belangrijk*

Van objectgeoriënteerde applicaties wordt gezegd dat de eerste coderegel de moeilijkste is om te schrijven. Dit gaat inderdaad op voor ObjectPAL, niet vanwege de inherente moeilijkheidsfactor van de taal, maar omdat de planning vooraf zo'n belangrijke fase is in het ontwikkelingsproces. Voordat u één enkele coderegel schrijft, moet u al een duidelijk idee hebben van wat de applicatie moet doen, welke objecten de applicatie gebruikt en hoe de gebruiker met deze objecten communiceert. Hoe ingewikkelder uw applicatie is, hoe belangrijker deze fase wordt. Als u een Paradox-applicatie wilt ontwikkelen, gaat u als volgt te werk:

1. Stel een doel.
2. Maak tabellen.

3. Ontwerp het formulier.
4. Koppel code.

## Een doel stellen

Als u een applicatie maakt, moet u uzelf eerst een doel stellen. Denk daarbij aan de gebruikers: wat willen zij doen? Begin met een algemeen antwoord. Een gebruiker wil bijvoorbeeld informatie verwerken over klanten en bestellingen. Het algemene doel van de applicatie is dan de gebruiker de mogelijkheid te bieden gegevens in te voeren en te bewerken en relevante gegevens op te zoeken.

Als u een grote applicatie maakt, is het mogelijk dat u het doel niet beknopt kunt formuleren. Een goede benadering is dan te analyseren wat u wilt doen en verscheidene subdoelen te definiëren. Een subdoel kan leiden tot een eigen formulier of een eigen pagina in een multi-pagina formulier. Paradox en ObjectPAL zijn ideale hulpmiddelen voor het maken van dit soort modulaire applicaties.

## Tabellen maken

De volgende stap is tabellen te maken die de gegevens bevatten, tenzij u al over deze benodigde gegevens beschikt. Vaak komen beide situaties voor. Als u bijvoorbeeld een applicatie voor de invoer van bestelinformatie maakt, hebt u misschien al tabellen met gegevens over klanten, maar daarnaast moet u tabellen maken om de bestelinformatie op te slaan. U kunt gegevens gebruiken die in verschillende formaten zijn opgeslagen, bijvoorbeeld als een tekstbestand of als spreadsheet-gegevens, maar als u de kracht van Paradox volledig wilt benutten, slaat u de gegevens in een tabel op.

Als uw applicatie meer dan één tabel gebruikt, moet u een gegevensmodel opstellen. Dit wil zeggen dat u bepaalt wat de onderlinge relatie van de tabellen is en welke velden u gebruikt om de tabellen te koppelen.

## Het formulier ontwerpen

Als u een doel hebt gesteld en tabellen hebt gemaakt, kunt u een formulier (of formulieren) ontwerpen, waarmee de gebruiker kan werken. Beslis welke objecten u hiervoor wilt gebruiken (knoppen, velden, tabelframes, multi-record objecten en wat de gebruiker nog meer nodig heeft om zijn werk te kunnen doen). Hoe meer u weet over de kenmerken en de mogelijkheden van deze objecten, des te beter kunt u ontwerpbeslissingen nemen.

***Belangrijk*** Als u alle objecten hebt geplaatst, start u het formulier en kijkt u hoe de objecten zich standaard gedragen. Daarna kunt u code koppelen. In de meeste gevallen zal het standaardgedrag overeenkomen met uw wensen. U hoeft alleen code aan een object te koppelen als u wilt dat het object iets anders doet.

## Code koppelen

Elk ontwerpobject bevat ingebouwde code. Als u objecten in een formulier plaatst, programmeert u dus eigenlijk. Alleen als een object iets anders (of iets meer) moet doen, koppelt u eigen ObjectPAL-code. In dit stadium is meer planning nodig, omdat u de volgende belangrijke vraag moet beantwoorden:

Waar plaats ik mijn code?

Voordat u antwoord kunt geven op deze vraag, moet u eerst twee andere vragen beantwoorden:

1. Wanneer moet de code worden uitgevoerd?
2. Hoeveel objecten gebruiken deze code?

### *Wanneer moet de code worden uitgevoerd?*

Voordat u deze eerste vraag kunt beantwoorden, moet u weten wanneer ingebouwde methodes van een object worden uitgevoerd. Ingebouwde methodes worden uitgevoerd in antwoord op acties. Wat is een *actie*? Een actie is een bericht aan een object dat door een bepaalde handeling wordt gegenereerd. Hier volgen enkele voorbeelden van handelingen die acties genereren: op de muisknop drukken, de muisknop loslaten, de aanwijzer op een object plaatsen, op een toets drukken, de invoegpositie in een veld plaatsen, de invoegpositie uit een veld weghalen, een optie kiezen in een menu—kortom alles wat u in Paradox doet, genereert een actie.

*Ingebouwde methodes worden uitgevoerd in antwoord op acties.*

Als u met een object communiceert, genereert u een actie. Paradox antwoordt op acties door methodes aan te roepen. Duidelijker gezegd: als een handeling een actie genereert, verzamelt Paradox gegevens over de actie, bepaalt het programma welk object het doel van de actie was en worden de gegevens naar dat object gestuurd. De gegevens over de actie activeren een van de ingebouwde methodes van het doelobject en de code van die methode wordt uitgevoerd.

Als u bijvoorbeeld wilt dat er iets gebeurt als de gebruiker op een knop drukt, koppelt u uw code aan de ingebouwde **pushButton**-methode van de knop. Als er iets moet gebeuren als de gebruiker de aanwijzer in een kader plaatst, koppelt u uw code aan de ingebouwde **mouseEnter**-methode van dat kader. Als er iets moet gebeuren als de gebruiker een waarde in een veld verandert, koppelt u uw code aan de ingebouwde **changeValue**-methode van het veld. (Alle ingebouwde methodes, met inbegrip van **pushButton**, **mouseEnter** en **changeValue**, worden beschreven in Appendix B.

U hoeft geen methodes te schrijven voor alle acties die een object kan behandelen. Alle objecten hebben ingebouwde methodes voor ObjectPAL-acties en acties worden altijd opgemerkt. Als u eigen code toevoegt, kunt u opgeven wanneer (en of) de ingebouwde code wordt uitgevoerd. Met ObjectPAL kunt u methodes schrijven die acties

behandelen, methodes schrijven die acties doorgeven aan andere objecten, en methodes schrijven die beide doen.

### *Hoeveel objecten gebruiken deze code?*

Als u deze tweede vraag wilt beantwoorden, moet u weten wat ingesloten objecten zijn (zie Hoofdstuk 11). Tabel 8-1 geeft een aantal algemene richtlijnen.

Tabel 8-1 Richtlijnen voor het koppelen van code aan objecten

| Objecten die de code gebruiken | Waar de code te koppelen |
|---|---|
| Eén object, één methode | Het object |
| Eén object, veel methodes | Een eigen methode of procedure koppelen aan het object |
| Veel objecten op hetzelfde formulier | Een eigen methode of procedure koppelen aan een insluitend object |
| Veel objecten, veel formulieren | In een bibliotheek |

Hoe weet u nu welk type en welke methode u moet gebruiken? Stel uzelf de vraag: "Met welk type *object* werk ik?" Elk object heeft een type. Zelfs eenvoudige objecten, zoals tekenreeksen, datums en getallen hebben types (respectievelijk, String, Date en Number).

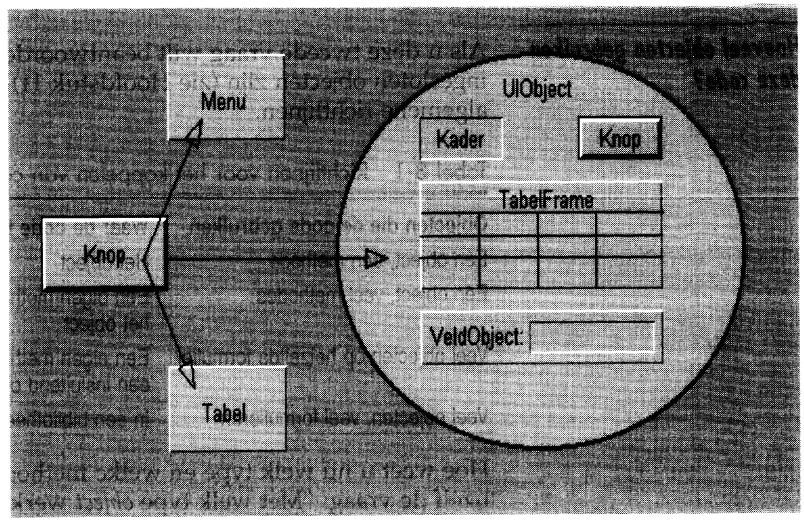
Meestal komen de naam van een object en het objecttype overeen: methodes van het type Table werken op tabellen, methodes van het type Menu werken op menu's enzovoort. Het UIObject-type wijkt hiervan af. Dit type bevat methodes die werken op knoppen, kaders, velden, tabelframes, multi-record objecten en op de andere objecten die u kunt maken met de hulpmiddelen op de TurboBalk.

**Opmerking** Pagina's op een formulier zijn UIObjecten en het formulier zelf kan zich gedragen als een UIObject. Zie Hoofdstuk 13 voor meer informatie en een volledig overzicht van UIObjecten.

*Het type object dat u bewerkt, bepaalt welke methodes u gebruikt.*

Als u moet beslissen welke methode u gebruikt, is het belangrijk onderscheid te maken tussen het object waaraan de methode is gekoppeld, en het object waarop de methode werkt. Zo is een knop een UIObject, maar u kunt er methodes aan koppelen die werken op objecten van elk ObjectPAL-type. Het type object dat u bewerkt, bepaalt welke methode u gebruikt. Als u een tabel wilt bewerken, gebruikt u Table-methodes. Voor een menu gebruikt u Menu-methodes, voor een multi-record object, een veld, een kader of een knop gebruikt u UIObject-methodes (dit zijn UIObjecten).

Afbeelding 8-4 Welke methode? Welk object?




De objecten waarmee u werkt, bepalen welke methodes u gebruikt. Code die is gekoppeld aan een knop, kan bijvoorbeeld op een ander object werken, maar de code moet methodes gebruiken die geschikt zijn voor het type object waarop deze werkt.

# Objectcategorieën

In deze handleiding worden types ingedeeld in de categorieën uit Tabel 8-2.

Tabel 8-2 Categorieën en objecttypes

| Categorie | Types |
|---|---|
| Acties | ActionEvent, ErrorEvent, Event, KeyEvent, MenuEvent, MouseEvent, MoveEvent, StatusEvent, TimerEvent, ValueEvent |
| Ontwerpobjecten | Menu, PopUpMenu, UIObject |
| Weergavebeheer-objecten | Application, Form, Report, TableView |
| Gegevenstypes | AnyType, Array, Binary, Currency, Date, DateTime, DynArray, Graphic, Logical, LongInt, Memo, Number, OLE, Point, Record, SmallInt, String, Time |
| Gegevensmodel-objecten | Database, Query, Table, TCursor |
| Systeemgegevens-objecten | DDE, FileSystem, Library, Session, System, TextStream |

Afbeelding 8-5 laat zien hoe objecttypes in categorieën zijn ingedeeld en hoe de categorieën zich ten opzichte van elkaar verhouden in een ObjectPAL-applicatie.

## Acties

De objecttypes die omgaan met *acties* staan boven in de afbeelding, omdat acties ObjectPAL activeren. Een gebruiker die met een applicatie communiceert, genereert acties. Methodes die aan objecten zijn gekoppeld, worden uitgevoerd in antwoord op deze acties.

Als u zich afvraagt hoe iets abstracts als een actie een object kan zijn, bedenk dan dat een actie bestaat uit gegevens en code en dus voldoet aan de definitie van een object. De gegevens bevatten informatie over het soort actie (bijvoorbeeld een muisklik of een toetsaanslag), over wat er is gebeurd (bijvoorbeeld welke muisknop of welke toets werd ingedrukt), over waar het is gebeurd (welk object was het doelobject) en over hoe het is gebeurd (deed de gebruiker bijvoorbeeld iets of is de actie vanuit ObjectPAL gegenereerd). De code is de ObjectPAL-methode voor het verzamelen van de gegevens.

## Ontwerpobjecten

De *ontwerpobjecten* bevinden zich op het volgende niveau, omdat ontwerpobjecten de code bevatten die wordt uitgevoerd in antwoord op acties. Ontwerpobjecten zijn de menu's, de pagina's, de knoppen, de kaders, de tabelframes en de andere objecten op een formulier, met inbegrip van het formulier zelf, waarmee de gebruiker communiceert (Afbeelding 8-5 toont niet alle objecten). De types in deze categorie zijn UIObject, Menu, en PopUpMenu. U kunt alleen aan UIObjecten code koppelen.

## Weergavebeheer-objecten

Daarna komen de *weergavebeheer-objecten*, omdat weergavebeheer-objecten ontwerpobjecten insluiten. Weergavebeheer-objecten zijn vensters, zoals formulieren en rapporten, die gegevens tonen. Het verschil tussen formulieren en rapporten is dat u geen code kunt koppelen aan rapporten. Methodes voor het TableView-type geven u de controle over het tabelvenster en methodes voor het Application-type geven controle over het bureaublad van Paradox.

## Gegevenstypes

Op het volgende niveau komen de elementaire *gegevenstypes* van ObjectPAL. Met deze gegevenstypes kunt u variabelen declareren om gegevens in tabellen op te slaan en te manipuleren. U kunt deze elementaire gegevenstypes natuurlijk ook gebruiken om berekeningen en bewerkingen uit te voeren zonder de gegevens uit tabellen te halen of de resultaten aan de gebruiker te tonen.

## Gegevensmodel-objecten

Daarna komen de *gegevensmodel-objecten*, omdat deze de gegevens verwerken die zijn opgeslagen in tabellen. De types Table, Query en TCursor geven toegang tot de gegevens en manipuleren deze. Het Database-type manipuleert verzamelingen tabellen.

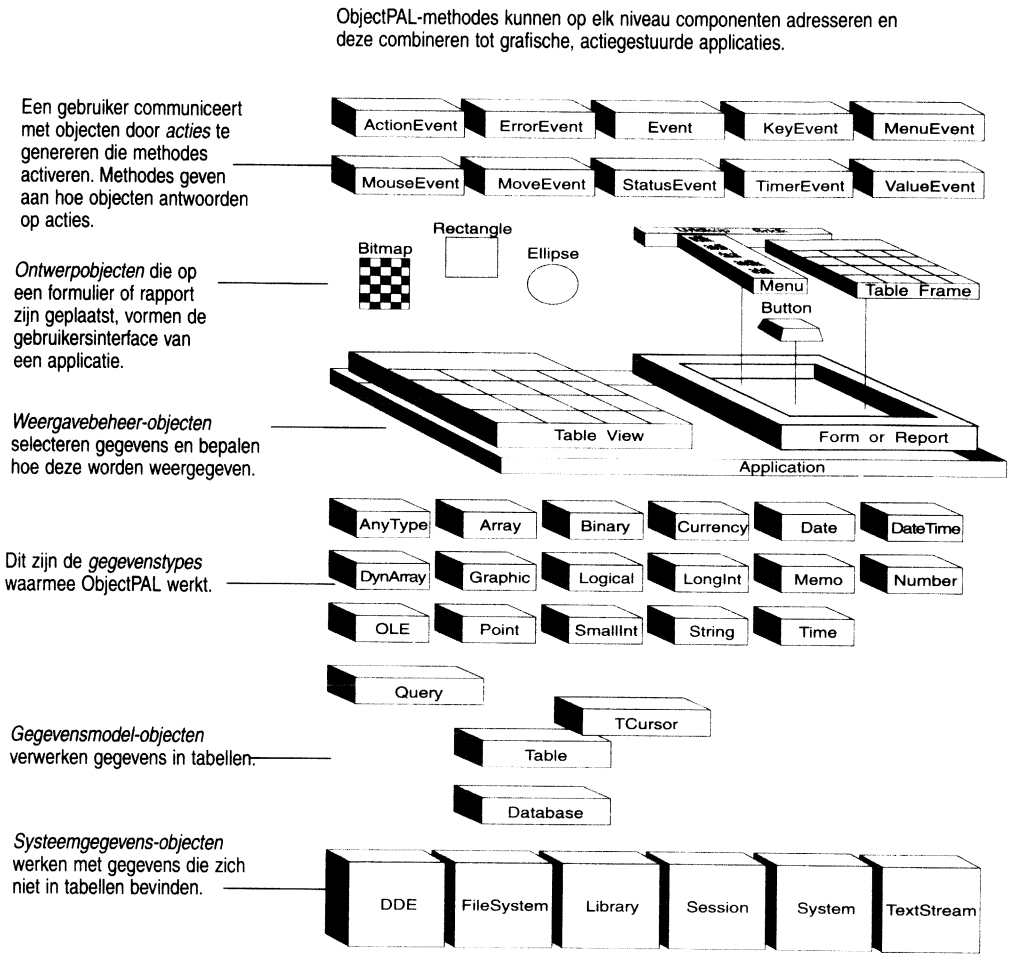
## Systeemgegevens-objecten

De *systeemgegevens-objecten* staan onder in de afbeelding. Deze objecten zorgen voor de opslag van en de toegang tot gegevens over Windows, DOS, aantallen gebruikers en het computersysteem van de gebruiker, maar *niet* over gegevens die in tabellen zijn opgeslagen.

*Samenvatting*

Als u een Paradox-applicatie gebruikt, genereert u acties door te communiceren met ontwerpobjecten die zich in weergavebeheer-objecten bevinden. Ontwerpobjecten en weergavebeheer-objecten vormen samen de interface van een applicatie. U kunt gegevenstypes gebruiken om variabelen te declareren die berekeningen uitvoeren en gegevens manipuleren in tabellen, en systeemgegevens-objecten voor informatie op systeemniveau. ObjectPAL kan op elk niveau met objecten werken.

Afbeelding 8-5 Componenten van een applicatie




# En nu?

Hoewel het aan te raden is de hoofdstukken in deze handleiding in de juiste volgorde te lezen, vooral als u geen ervaring hebt met actiegestuurde programmering, kunnen ervaren programmeurs bepaalde gedeelten overslaan. Deze paragraaf verwijst naar de gedeelten in de documentatie, waarin wordt uitgelegd hoe u veel voorkomende programmeertaken uitvoert. Bekijk ook de voorbeeldbestanden, de index en de online Helpfuncties.

## Tabellen

Voordat u ObjectPAL gebruikt om tabellen te manipuleren, moet u iets weten van de types Table, TCursor en TableView en moet u de volgende UIObjecten kennen: tabelframe, multi-record object en veldobject. Tabel 8-3 geeft een overzicht van de functies van deze objecten.

Tabel 8-3 Objecten voor het werken met tabellen

| Type | Beschrijving |
|---|---|
| Table | Een beschrijving van een tabel: lokatie, structuur, type enzovoort. |
| TCursor | Een verwijzing naar een tabel die wordt opgeslagen en gemanipuleerd in het geheugen. |
| TableView | Een tabel die is weergegeven in een eigen venster. |
| UIObject | Een tabelframe geeft een tabel weer in rijen en kolommen. Een multi-record object geeft velden uit één of meer records van een tabel weer. Een veldobject geeft gegevens uit één veld van een tabel weer. |

Raadpleeg de volgende paragrafen voor meer informatie over deze objecttypes:

- Raadpleeg het *Handboek*, voor informatie over hoe u een tabel maakt en structureert, en hoe u een tabelframe in een formulier plaatst en het met een tabel verbindt.
- Tabellen en TCursors worden behandeld in Hoofdstuk 16.
- UIObjecten, zoals tabelframes, multi-record objecten en veldobjecten, worden behandeld in Hoofdstuk 13.
- TableViews worden behandeld in Hoofdstuk 14.
- De volgende voorbeeldapplicaties:
  - Duikplanning (MAST)
  - Adressen
  - Giro
  - Eenarm

## Queries

U kunt ObjectPAL gebruiken voor het maken en starten van queries waarvoor u zelf alle code hebt geschreven en queries die u interactief met Paradox hebt gemaakt.

- De methodes voor het werken met queries worden beschreven in Hoofdstuk 16.
- De Adressen-applicatie geeft een voorbeeld van een manier om query-resultaten in een formulier weer te geven.

## Berichten en dialoogvensters

Berichten en ingebouwde dialoogvensters zijn middelen om te communiceren met een gebruiker. Zie de volgende paragrafen:

- Informatie over **msgAbortRetryIgnore**, **msgInfo**, **msgQuestion**, **msgRetryCancel**, **msgStop**, **msgYesNoCancel** in de paragraaf "System" van Hoofdstuk 17
- De paragraaf "Multi-formulier applicaties" in Hoofdstuk 14

## Toetsenbordacties

U kunt in ObjectPAL reageren op elke toetsaanslag. Dit wil zeggen dat u eenvoudig sneltoetsen kunt ontwikkelen voor uw applicatie. Zie de volgende paragrafen en appendixen:

- De beschrijving van het actiemodel in Hoofdstuk 12
- **keyPhysical** en **keyChar** in Appendix B
- De voorbeeldapplicatie Giro

## Muisacties

U kunt ObjectPAL gebruiken om alle muishandelingen te behandelen: klikken, dubbelklikken, verplaatsingen enzovoort. Zie de volgende hoofdstukken en paragrafen:

- De paragraaf "MouseEvent" in Hoofdstuk 12
- De volgende voorbeeldapplicaties:
  - Duikplanning
  - Adressen
  - Paradox Teken

## Menu's

Met ObjectPAL kunt u menu's en pop-up menu's definiëren die opties tonen aan gebruikers. Zie verder de volgende paragrafen:

- De paragraaf "MenuEvent" in Hoofdstuk 12
- "Menu" en "PopUpMenu" in Hoofdstuk 13
- De volgende voorbeeldapplicaties:
  - Duikplanning
  - Eenarm

## Lijsten

U kunt keuzelijsten en afrollijsten gebruiken om een gebruiker te laten kiezen uit een aantal opties. Plaats een veldobject en stel het weergavetype in op 'Lijst' of 'Afrol-bewerken'. Als u wilt weten hoe u deze lijsten manipuleert, raadpleeg dan:

- "Tabellen en TCursors gebruiken voor het programmeren van lijsten" in Hoofdstuk 16

## Multi-formulier applicaties

Als u applicaties wilt ontwerpen die meer dan één formulier gebruiken, moet u weten hoe u een formulier opent en het bestuurt vanuit een ander formulier. U kunt formulieren ook openen als dialoogvensters. Zie de volgende paragrafen:

- "Form" en "Multi-formulier applicaties" in Hoofdstuk 14

## Tekst

U kunt de volgende ObjectPAL-types gebruiken om met tekst te werken: String (gewone tekst), Memo (opgemaakte tekst) en TextStream (tekstbestanden). Zie de volgende paragrafen:

- "String" in Hoofdstuk 15
- "Memo" in Hoofdstuk 15
- "TextStream" in Hoofdstuk 17

## Het bestandssysteem

Methodes van het FileSystem-type geven toegang tot en verschaffen informatie over schijfbestanden, stations en directories. ObjectPAL bevat ook een ingebouwd dialoogvenster 'Bladermodus'. Zie de volgende paragrafen voor informatie over de FileSystem-methodes:

- "FileSystem" in Hoofdstuk 17
- De beschrijving van de **fileBrowser**-procedure van het System-type in Hoofdstuk 17

## Codebibliotheken

U kunt eigen ObjectPAL-code opslaan in bibliotheken en deze beschikbaar maken voor een of meer formulieren of applicaties. Zie de volgende paragrafen voor informatie over bibliotheken:

- De paragraaf "Library" in Hoofdstuk 17
- De ObjectPAL Helptekst bij het taalelement **uses**

## DLL's

Met **uses** kunt u functies uit DLL's (Dynamic Link Libraries) declareren en deze vervolgens aanroepen. Als u wilt weten hoe u een DLL koppelt, raadpleegt u:

- De ObjectPAL Helptekst bij het taalelement **uses**

HOOFDSTUK 9

# De ObjectPAL-Editor

Net als Kladblok heeft de ObjectPAL-Editor heeft de functies van een Windows-tekstverwerker, met daarnaast een aantal speciale functies (zoals syntaxiscontrole) voor de bewerking van ObjectPAL-methodes. In de ObjectPAL-Editor kunt u dialoogvensters openen die overzichten geven van de methodes van elk objecttype, van de syntaxis van de methodes, van de kenmerken en kenmerkwaarden van de objecten en van constanten voor zaken als vensterattributen en foutcodes. De Editor is nauw verbonden met de ObjectPAL De compiler vertaalt de geschreven ObjectPAL-code in machinecode, zodat een computer de code kan uitvoeren. Als u de Editor gebruikt, kan de compiler uw code controleren en fouten melden, zodat u deze kunt verbeteren voordat u de applicatie uitprobeert.

De Editor werkt ook met de Debugger (zie het volgende hoofdstuk), waardoor u een geïntegreerde omgeving tot uw beschikking hebt voor het maken, testen en wijzigen van methodes. U kunt uw methodes bewerken, fouten uit uw code halen en de objecten inspecteren om de methodes, de sleutelwoorden, de handelingen en de constanten te bekijken die ObjectPAL ondersteunt.

*U kunt een andere editor gebruiken.*

U kunt als u wilt een andere editor gebruiken (zie de bespreking van het menu 'Kenmerken', verderop in dit hoofdstuk).

Dit hoofdstuk behandelt de volgende onderwerpen:

- De Editor starten
- Werken met de Editor
- De Editor verlaten
- Snelle manieren

# Editor starten

U start de Editor, door een object te inspecteren en vervolgens 'Methodes' te kiezen uit de kenmerkenlijst. Het methodevenster (Afbeelding 9-1) geeft een overzicht van de methodes die u kunt bewerken. Het methodevenster bevat ook de elementen 'Var' (voor de declaratie van variabelen), 'Const' (voor de declaratie van constanten), 'Type' (voor de declaratie van gegevenstypes), 'Procs' (voor de declaratie van procedures) en 'Uses' (voor de declaratie van externe routines). Deze elementen hebben elk een eigen Editor-venster. U opent het Editor-venster door het bijbehorende element te kiezen en vervolgens 'OK' te kiezen. U kunt, in elke gewenste volgorde, zoveel vensters openen als het systeem toestaat.

***Snelle manier***

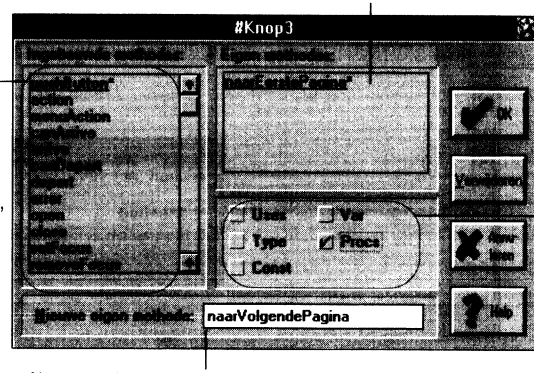
U opent het methodevenster door een object te selecteren en op *Ctrl-Spatiebalk* te drukken.

Kies een methode (bijvoorbeeld **open**) en kies vervolgens 'OK' (of dubbelklik alleen op de methode). Er verschijnt een Editor-venster dat lijkt op het venster uit Afbeelding 9-2 met wat standaardtekst.

*Er kunnen meerdere Editor-vensters geopend zijn.*

U bewerkt elke methode in een eigen venster. U kunt echter meerdere methodes tegelijk bewerken door meerdere vensters te openen. U selecteert hiervoor de methodes die u wilt bewerken, en kiest vervolgens 'OK'. Als u meer dan één methode wilt selecteren, drukt u op *Shift* of *Ctrl* en klikt u. U kunt ook in het methodevenster met de muis slepen. Als u 'OK' kiest, opent de Editor een venster voor elke methode die u hebt geselecteerd. Als u bijvoorbeeld de methodes **open** en **close** selecteert, verschijnen er twee vensters als u 'OK' kiest, één voor elke geselecteerde methode.

Afbeelding 9-1 Het methodevenster



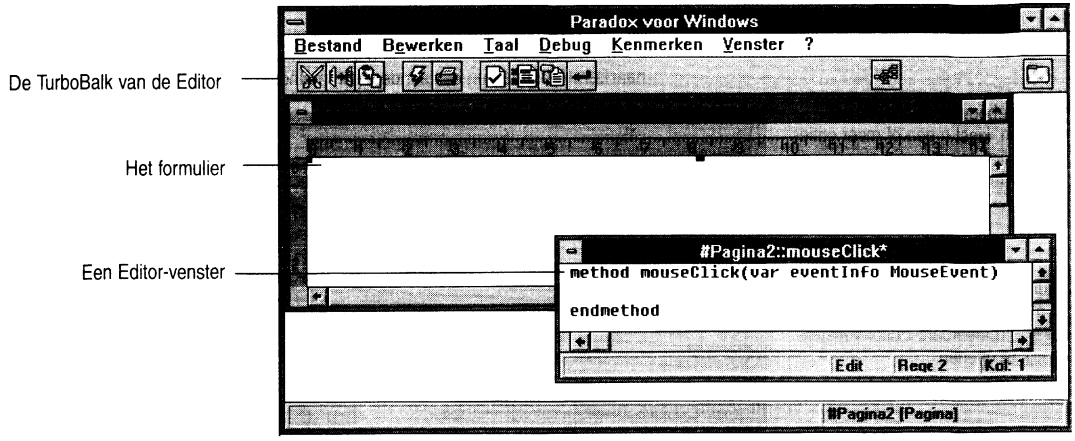

Als u de **open**-methode bewerkt, bevat de eerste regel de tekst **method open (var eventInfo Event)**, de tweede regel is leeg en de derde regel bevat het woord **endmethod**. Als u de standaardtekst per ongeluk verandert, kunt u deze als elke andere tekst bewerken.

*In de Editor is altijd de invoegmodus actief.*

Als u een Editor-venster opent om een methode te bewerken, bevindt de invoegpositie zich op regel 2. U kunt dus gelijk beginnen te typen, maar u *hoeft* niet op die regel te beginnen. U kunt de muis of de pijltoetsen gebruiken om de invoegpositie te verplaatsen en u kunt lege regels invoegen door op *Enter* te drukken.

***Opmerking*** Variabelen, constanten, en procedures die in een methodevenster zijn gedeclareerd, zijn alleen voor die methode zichtbaar. Als u een variabele, constante of procedure zichtbaar wilt maken voor alle methodes van een object, moet u een apart venster openen. Zo bepaalt u het bereik van een variabele. Zie Hoofdstuk 11 voor meer informatie over het bereik van variabelen.

Afbeelding 9-2 De Editor




# Werken met de Editor

De volgende paragrafen beschrijven hoe u de Editor gebruikt met de Editor-menu's en het toetsenbord.

Deze paragraaf beschrijft de opties in de Editor-menu's, de speciale Editor-toetsen en de manier waarop u tekst selecteert.

Als u een Editor-venster opent, verandert de applicatiemenubalk en geeft deze functies weer voor de bewerking van code. Deze functies worden in de volgende paragrafen beschreven.

***Opmerking*** Zie de paragraaf "Snelle manieren", verderop in dit hoofdstuk, voor een beschrijving van de TurboBalk-knoppen en de sneltoetsen die u in de Editor en de Debugger kunt gebruiken.

## Bestand

De werking van de opties in het menu 'Bestand' wordt beschreven in het *Handboek*. U kunt 'Bestand | Afdrukken' kiezen om de inhoud van het actieve Editor-venster of het actieve Traceervenster af te drukken.

## Bewerken

Deze paragraaf behandelt de opties in het menu 'Bewerken'.

*Ongedaan maken* Maakt uw laatste bewerking ongedaan.

*Knippen* Kopieert geselecteerde tekst naar het Klembord en verwijdert deze uit het venster. 'Knippen' is niet beschikbaar als er niets is geselecteerd.

*Kopiëren* Kopieert geselecteerde tekst naar het Klembord. 'Kopiëren' is niet beschikbaar als er niets is geselecteerd.

*Plakken* Kopieert tekst uit het Klembord naar de invoegpositie. Als er tekst is geselecteerd, wordt deze vervangen door de inhoud van het Klembord. 'Plakken' is niet beschikbaar als het Klembord geen tekst bevat.

*Verwijderen* Verwijdert geselecteerde tekst. 'Verwijderen' is niet beschikbaar als er geen tekst is geselecteerd.

*Plakken uit bestand* Biedt u de mogelijkheid een tekstbestand op te geven dat u op de invoegpositie wilt plakken.

*Kopiëren naar bestand* Kopieert geselecteerde tekst, en *niet* de inhoud van het Klembord van Windows, naar een tekstbestand dat u opgeeft.

*Zoeken* Doorzoekt de tekst vanaf de invoegpositie tot het einde van de methode of vanaf de invoegpositie tot het begin. U kunt het onderscheid tussen hoofdletters en kleine letters selecteren of deselecteren door op een aankruisvak te klikken.

*Volgende zoeken* Zoekt verder naar opgegeven tekst. 'Volgende zoeken' is niet beschikbaar als u nog niets hebt gezocht in het venster.

*Vervangen* Zoekt tekst en vervangt deze door de opgegeven vervangende waarde.

*Volgende vervangen* Zoekt verder naar tekst die is opgegeven met 'Vervangen', en vervangt deze. 'Volgende vervangen' is pas beschikbaar als er tekst is vervangen.

*Gaan naar* Gaat naar een opgegeven regel. Kies 'Bewerken | Gaan naar' om een dialoogvenster te openen, voer een regelnummer in en kies 'OK'. Als het regelnummer niet bestaat, verandert de invoegpositie niet van plaats.

*Alles selecteren* Selecteert alle tekst in het venster.

## Taal

Deze paragraaf beschrijft de opties in het menu 'Taal'.

*Syntaxis controleren* Controleert de syntaxis van de code in het actieve Editor-venster. Als er syntaxisfouten worden gevonden, verschijnt een foutmelding op de statusregel. Als u de syntaxis van alle code in het formulier wilt controleren, moet u het formulier starten of opslaan.

***Opmerking*** Als u code verandert in meer dan één Editor-venster, moet u de veranderingen opslaan voordat u de syntaxis controleert. Anders kan de syntaxiscontrole onverwachte fouten aangeven, omdat niet de meest recente code wordt gecontroleerd.

*Volgende waarschuwing* Toont de volgende waarschuwing die door de compiler is gevonden.

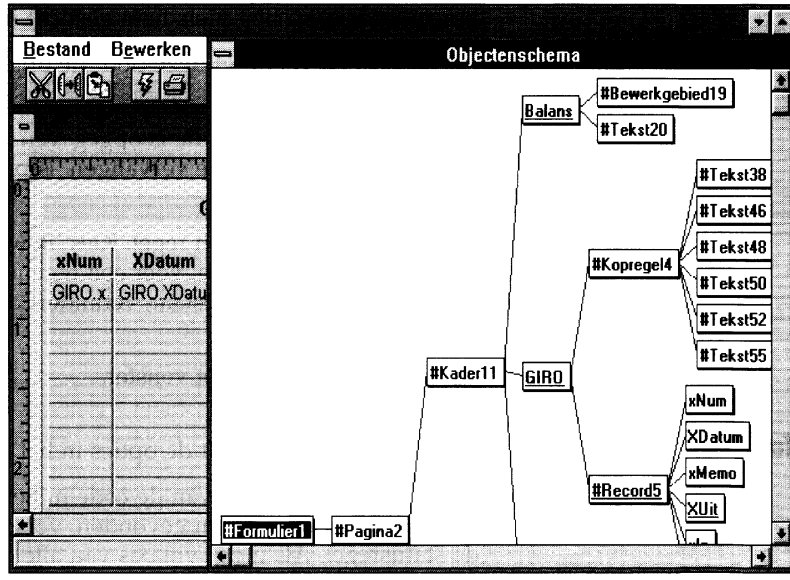
*Methodes* Opent het methodevenster.

*Objectenschema*

Geeft een schema weer dat de relaties toont tussen objecten van het huidige formulier. Het schema geeft de objecthiërarchie weer. Het geselecteerde object bevindt zich helemaal links in het schema en de hiërarchie van ingesloten objecten loopt naar rechts door (zie Afbeelding 9-3).

Als u een object op een formulier plaatst, krijgt het een standaardnaam die begint met een nummerteken (#). Het Objectenschema toont objecten die u hebt geplaatst en benoemd en objecten die u hebt geplaatst maar niet benoemd. Als u methodes hebt geschreven voor een object, wordt de naam van het object onderstreept. U kunt een object inspecteren in het Objectenschema en 'Methodes' kiezen om het bijbehorende methodevenster te openen. U kunt de muis of de pijltoetsen gebruiken om de markering te verplaatsen in het Objectenschema.

Afbeelding 9-3 Het Objectenschema

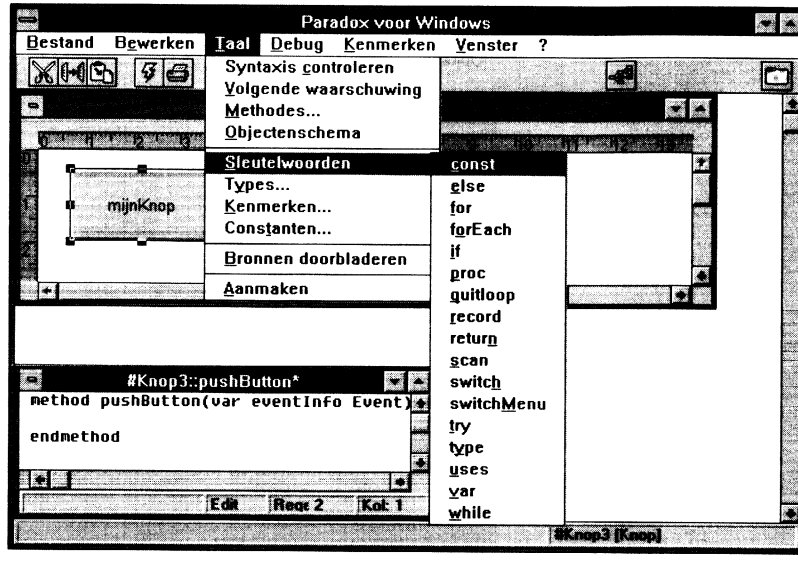



*Sleutelwoorden*

Opent een vervolgmenu met veelgebruikte woorden die door Paradox zijn gereserveerd. U gebruikt dit menu om een sleutelwoord in een methode in te voegen zonder het woord te hoeven typen. De sleutelwoorden zijn elementaire taalelementen en worden beschreven in de ObjectPAL online Help.

Afbeelding 9-4 toont het menu dat verschijnt als u 'Taal | Sleutelwoorden' kiest.

Afbeelding 9-4 Het menu 'Sleutelwoorden'

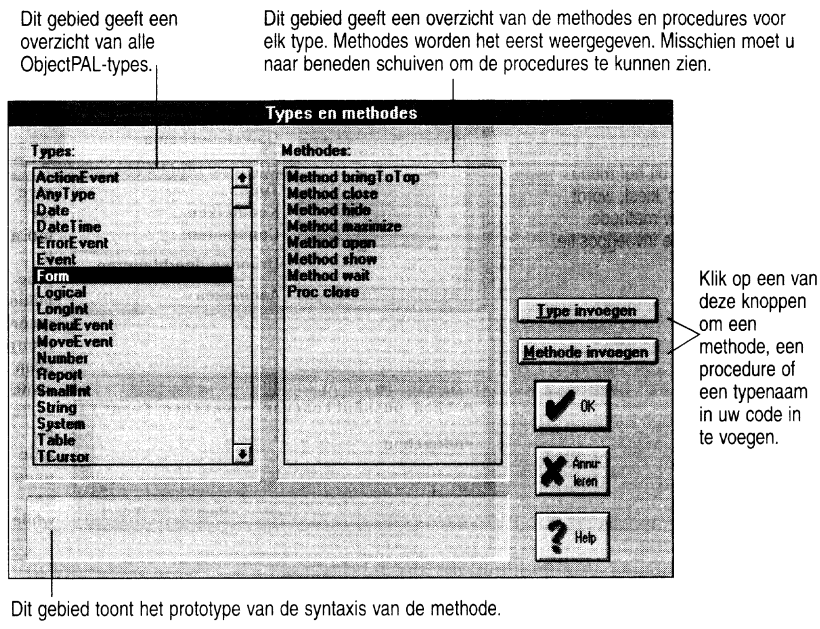



Als u een optie uit het menu 'Sleutelwoorden' kiest, wordt deze optie in uw methode ingevoegd op de invoegpositie.

*Types* Opent een dialoogvenster met een overzicht van alle objecttypes en de bijbehorende methodes en procedures. U kunt een typenaam kiezen om de methodes en procedures voor dat type te bekijken. De methodes worden eerst, in alfabetische volgorde, weergegeven, gevolgd door de procedures, eveneens in alfabetische volgorde. U moet misschien door het overzicht schuiven om alle procedures te kunnen zien. U kunt een prototype van de methode of procedure in uw eigen methode invoegen op de invoegpositie. Als u bijvoorbeeld een overzicht wilt zien van de methodes en procedures voor het Array-type, kiest u 'Array'.

Afbeelding 9-5 toont het dialoogvenster dat verschijnt als u 'Taal | Types' kiest.

Afbeelding 9-5 Het dialoogvenster 'Types en methodes'

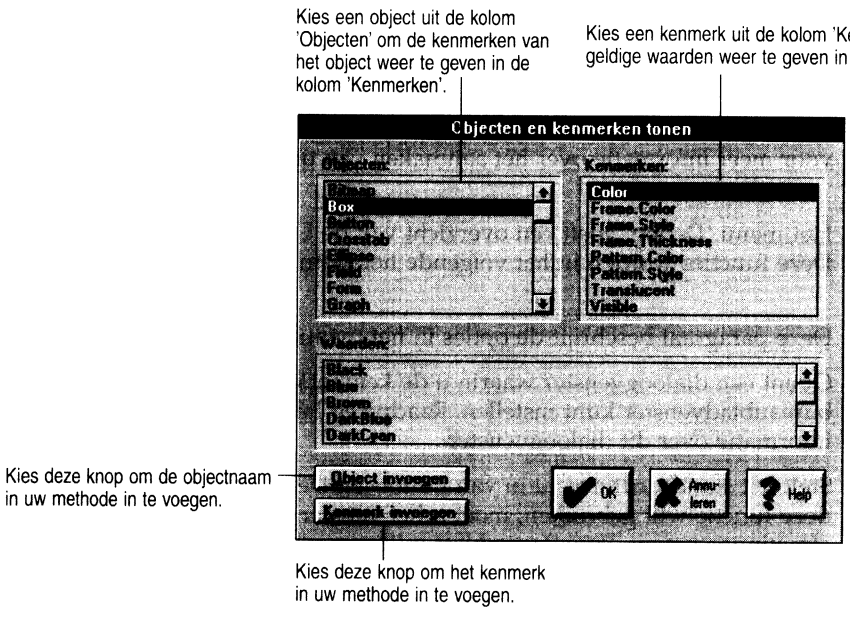



*Kenmerken*

Opent een dialoogvenster met een overzicht van objecten en hun kenmerken. U kunt een objectnaam kiezen om de kenmerken voor dat object te bekijken. Vervolgens kunt u het kenmerk invoegen in uw methode. Als u bijvoorbeeld een overzicht wilt zien van de kenmerken van 'Button', kiest u 'Button'.

Afbeelding 9-6 toont het dialoogvenster dat verschijnt als u 'Taal | Kenmerken' kiest.

Afbeelding 9-7 Het dialoogvenster 'Objecten en kenmerken tonen'

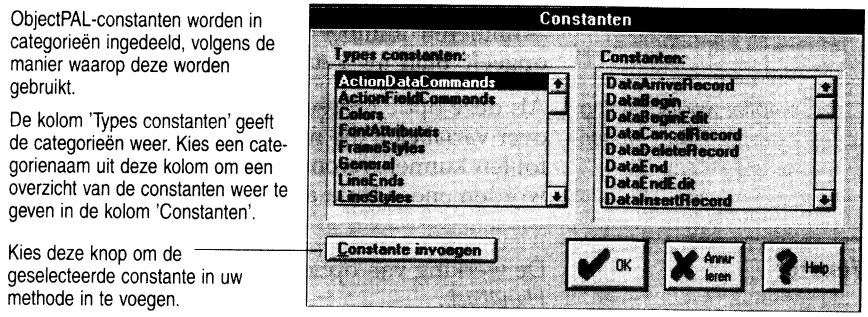



*Constanten*

Opent een dialoogvenster met een overzicht van ObjectPAL-constanten. ObjectPAL beschikt over veel constanten, waardoor u gemakkelijk allerlei dingen kunt opgeven, zoals kleuren, muisvormen, menu-attributen en vensterstijlen. U kunt een constante kiezen en vervolgens 'Constante invoegen' kiezen om deze constante in uw code in te voegen.

Afbeelding 9-6 toont het dialoogvenster dat verschijnt als u 'Taal | Constanten' kiest.

Afbeelding 9-6 Het dialoogvenster 'Constanten'



*Bronnen doorbladeren* Maakt en toont een rapport met de broncode op het formulier of rapport waarmee u werkt. De gegevens van dit rapport worden opgeslagen in een tabel, genaamd PAL$SRC.DB, in uw privé-directory.

*Aanmaken* Maakt gecompileerde formulieren, bibliotheken of scripts aan, die zijn ontdaan van hun ObjectPAL-broncode. Andere gebruikers kunnen het ontwerp of de broncode dus niet bewerken. Zie verder Hoofdstuk 21 voor meer informatie over het aanmaken van objecten.

## Debug

Het menu 'Debug' geeft een overzicht van de Debugger-functies. Deze functies worden in het volgende hoofdstuk behandeld.

## Kenmerken

Deze paragraaf beschrijft de opties in het menu 'Kenmerken'.

*Bureaublad* Opent een dialoogvenster waarin u de kenmerken van het bureaubladvenster kunt instellen. Raadpleeg het *Handboek* voor meer informatie over dit dialoogvenster.

*Venstermaat aanpassen* Stelt het standaardformaat in van Editor- en Debugger-vensters. Als u deze functie wilt gebruiken, moet u het venster eerst de gewenste afmetingen geven. Vervolgens kiest u 'Kenmerken | Venstermaat aanpassen' om een dialoogvenster te openen dat twee opties bevat: **Gebruik voortaan huidig formaat** en **Terug naar standaardformaat**.

*Andere editor* Opent een dialoogvenster waarin u een andere editor kunt opgeven voor het schrijven en bewerken van methodes. (De editor die u gebruikt, moet alleen-tekst bestanden kunnen opslaan zonder opmaakcodes.) Typ vervolgens het volledige pad van de gewenste editor, bijvoorbeeld C:\APPS\BRIEF\B.EXE.

Als u een andere editor gebruikt, hebt u geen toegang tot de syntaxiscontrole of andere on-line informatie van de ObjectPAL-Editor, maar u kunt wel schakelen tussen editors.

Als u een andere editor opgeeft, verschijnt er telkens als u een methode bewerkt, een dialoogvenster. Als u 'Ja' kiest, opent u de andere editor en als u 'Nee' kiest opent u de ObjectPAL-Editor. Met 'Annuleren' annuleert u de handeling. Als u een andere editor opgeeft, blijft uw keuze van kracht totdat u Paradox verlaat.

*Compiler-waarschuwing tonen* Als deze optie is geselecteerd, verschijnt op de statusregel informatie over variabelen die niet zijn gedeclareerd, en andere situaties die fouten kunnen veroorzaken als u de applicatie start. Deze berichten worden onderdrukt als de menu-optie niet is geselecteerd.

## Venster

De werking van opties in het menu 'Venster' wordt beschreven in het *Handboek*.

## Help

De werking van opties in het menu 'Help' (?) wordt beschreven in het *Handboek*.

## Toetsenbord

In deze paragraaf wordt uitgelegd hoe u het toetsenbord gebruikt om de invoegpositie te verplaatsen en code te bewerken in een Editor-venster. Tabel 9-1, verderop in dit hoofdstuk, geeft een overzicht van andere sneltoetsen.

U kunt de invoegpositie in een venster met één teken of één regel tegelijk verplaatsen met behulp van de pijltoetsen. Gebruik *Ctrl* ← en *Ctrl* → om de invoegpositie met één woord tegelijk te verplaatsen.

*Home* verplaatst de invoegpositie naar het begin van een regel, terwijl *End* de invoegpositie naar het einde van de regel verplaatst. *Ctrl-Home* verplaatst de invoegpositie naar het begin van de tekst en *Ctrl-End* verplaatst de invoegpositie naar het einde van de tekst.

*PgUp* en *PgDn* verplaatsen de invoegpositie met één venster tegelijk.

*Backspace* verwijdert het teken links van de invoegpositie. *Del* verwijdert het teken rechts van de invoegpositie.

Op zichzelf doet *Ins* niets. In de ObjectPAL-Editor is altijd de invoegmodus actief. Als u typt, worden de tekens altijd naar rechts geschoven. U kunt niet over tekens heen typen. *Ctrl-Ins* kopieert geselecteerde tekst naar het Klembord, en *Shift-Ins* plakt tekst uit het Klembord in uw methode.

***Opmerking*** De ObjectPAL-Editor zorgt niet automatisch voor regelovergangen. Een regel breidt zich tijdens het typen naar rechts uit totdat u op *Enter* drukt om een nieuwe regel te beginnen.

Tab voegt een onzichtbaar tab-teken (ANSI 9) in en schuift de tekst naar rechts.

## Tekst selecteren

U kunt een tekstblok selecteren door met de muis te slepen, door de pijltoetsen te gebruiken terwijl u *Shift* ingedrukt houdt of door te klikken terwijl u *Shift* ingedrukt houdt om de selectie uit te breiden. U kunt een hele regel selecteren door links van de regel te klikken. (U kunt dit pas doen als de invoegpositie in een pijl verandert.) U selecteert een woord door erop te dubbelklikken.

Als u tekst hebt geselecteerd, wordt deze vervangen door tekst die u typt of uit het Klembord plakt.

# ObjectPAL-Editor verlaten

U kunt de Editor op verschillende manieren verlaten. Welke manier u kiest, hangt af van wat u vervolgens wilt doen.

- Als u verder wilt gaan met het ontwerp van uw formulier of met de bewerking van andere methodes, gaat u als volgt te werk:
  - Dubbelklik op het Systeemmenu van het Editor-venster of kies 'Sluiten' in het Systeemmenu.
  - Klik op de knop 'Bron opslaan en afsluiten' op de TurboBalk.
- Als u uw formulier wilt bekijken en het wilt zien werken, klikt u op de knop 'Gegevens tonen op de TurboBalk.
- Als u uw code meteen wilt starten, klikt u op de knop 'Formulier starten' of kiest u 'Debug | Starten'.

Als u wijzigingen hebt aangebracht in uw code, maar deze niet hebt opgeslagen, verschijnt er een dialoogvenster waarin u de wijzigingen kunt opslaan of kunt annuleren.

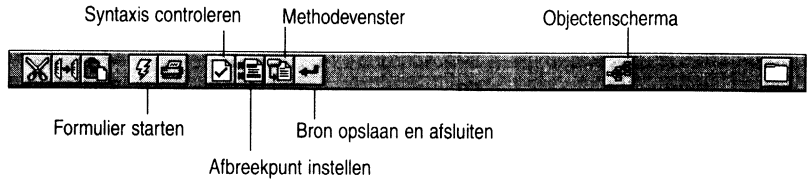
# Snelle manieren

De ObjectPAL-Editor bevat TurboBalk-knoppen en ondersteunt de rechtermuisknop als snelle manieren voor veel voorkomende handelingen. Afbeelding 9-8 toont de TurboBalk-knoppen van de Editor.

## TurboBalk-knoppen

- 'Formulier starten' heeft dezelfde werking als de optie 'Formulier | Gegevens tonen'.
- 'Syntaxis controleren' heeft dezelfde werking als de optie 'Taal | Syntaxis controleren'.
- 'Afbreekpunt instellen' heeft dezelfde werking als de optie 'Debug | Afbreekpunt instellen'.
- 'Methodevenster' heeft dezelfde werking als de optie 'Taal | Methodes'.
- 'Bron opslaan en afsluiten' heeft dezelfde werking als 'Sluiten' kiezen in het Systeemmenu van het venster.
- 'Objectenschema' heeft dezelfde werking als de optie 'Taal | Objectenschema'.

Afbeelding 9-8 De knoppen op de TurboBalk van de Editor




## Rechtermuisknop

Als u een methode bewerkt, kunt u rechts klikken om een pop-up menu te openen. Dit menu bevat dezelfde opties als het menu 'Taal'.

Als u fouten in een methode zoekt, klikt u rechts om een ander pop-up menu te openen. Dit menu bevat dezelfde opties als het menu 'Debug' (zie het volgende hoofdstuk voor meer informatie over het gebruik van de Debugger).

## Editor- en Debugger-sneltoetsen

Tabel 9-1 geeft een overzicht van de sneltoetsen die u kunt gebruiken in de Editor en de Debugger.

Tabel 9-1 Editor- en Debugger-sneltoetsen

| Functiesneltoets | Sneltoets | Functie |
|---|---|---|
| *F1* | | Help |
| *F2* | | Wijzigingen accepteren |
| *Shift-F2* | | Wijzigingen accepteren en venster sluiten |
| | *Ctrl-Z* | Zoeken |
| | *Ctrl-Shift+Z* | Zoeken en vervangen |
| | *Ctrl-A* | Volgende zoeken |
| *F3* | *Ctrl-I* | Variabele inspecteren (Debugger) |
| *Ctrl-F3* | *Ctrl-B* | Afbreekpunt instellen (Debugger) |
| *Shift-F3* | *Ctrl-K* | Stapel terugvolgen (Debugger) |
| *F5* | *Ctrl-G* | Gaan naar regel |
| *Ctrl-F5* | *Ctrl-W* | Volgende waarschuwing |
| *F6* | *Ctrl-M* | Taalmenu |
| *F7* | | Overheen stappen (Debugger) |
| *Shift-F7* | | Stappen in (Debugger) |

| Functiesneltoets | Sneltoets | Functie |
|---|---|---|
| *F8* | | Starten (Tot afbreekpunt indien in Debugger) |
| *Shift-F8* | | Formulier opslaan op schijf en starten |
| *F9* | *Ctrl-Y* | Syntaxis controleren (Compileren) |
| *Ctrl-F9* | *Ctrl-D* | Formulier aanmaken |
| | *Ctrl-Ins* | Kopiëren |
| | *Shift-Del* | Knippen |
| | *Shift-Ins* | Plakken |
| | *Ctrl-Spatiebalk* | Methodevenster openen |

HOOFDSTUK 10

# De ObjectPAL-Debugger

In dit hoofdstuk wordt uitgelegd hoe u de Debugger start, met de Debugger werkt en de Debugger verlaat. Aan het eind van het hoofdstuk wordt een voorbeeld van een Debugger-sessie beschreven die laat zien hoe u fouten opspoort.

***Opmerking*** In dit hoofdstuk wordt ervan uitgegaan dat u vertrouwd bent met de ObjectPAL-Editor, die in Hoofdstuk 9 is beschreven.

## Fouten? Ik?

Ook u kunt fouten maken. Computers staan erom bekend dat ze doen wat u zegt en niet per se wat u wilt. Als u deze kloof wilt overbruggen, moet u uw code ontdoen van fouten (*bugs*) die zorgen voor onverwachte resultaten. Dit proces wordt *debugging* genoemd.

De eerste stap bij het opsporen van fouten in een methode is erachter komen dat er een fout is. Soms is dit duidelijk: u start de applicatie voor de eerste keer en de fout steekt de kop op. Andere fouten zijn moeilijker op te sporen en kunnen lang een verborgen leven leiden totdat een methode een bepaalde waarde ontvangt (vaak 0 of een negatief getal) of totdat u de uitvoer eens goed bekijkt en ontdekt dat de resultaten met een factor van 0,2 afwijken of dat de middelste initialen in een lijst met namen niet kloppen.

Als u onverwachte resultaten ziet, is de volgende stap de fout op te sporen en deze te herstellen. Normaal gesproken worden de instructies in een methode te snel uitgevoerd om deze te kunnen volgen. Het is dus moeilijk te bepalen waar de dingen precies misgaan. In dit geval kan de Debugger uitkomst bieden. De Debugger biedt de volgende mogelijkheden:

- U kunt afbreekpunten instellen, zodat u instructies kunt uitvoeren tot aan een bepaald punt en daar kunt stoppen om te bekijken wat er tot dan toe is gebeurd.

- U kunt een venster openen dat elke coderegel weergeeft terwijl deze wordt uitgevoerd.
- U kunt variabelen inspecteren om te controleren of waarden op de juiste manier worden gemanipuleerd.
- U kunt een methode regel voor regel uitvoeren (*enkele-stapmethode* genoemd).
- U kunt methodes en procedures overslaan die geen fouten bevatten.

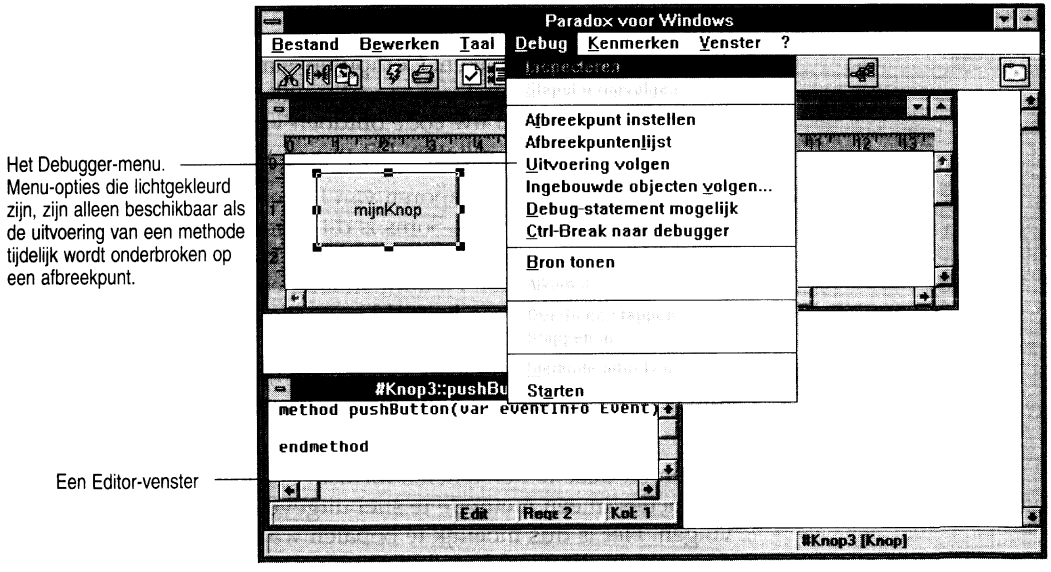
# Werken met de Debugger

Deze paragraaf beschrijft hoe u de Debugger gebruikt.

## Debugger starten

Open eerst een methode in een Editor-venster. Kies vervolgens 'Debug' om het afrolmenu met Debugger-functies te openen (zie Afbeelding 10-1).

Afbeelding 10-1 Het Debugger-menu




## Debug

Het menu 'Debug' biedt de volgende mogelijkheden:

- *Inspecteren* toont de waarde van een variabele en verandert deze eventueel. Deze opdracht is alleen beschikbaar als de uitvoering wordt stopgezet op een afbreekpunt.

- *Stapel terugvolgen* geeft een overzicht van de methodes en procedures die zijn aangeroepen sinds u uw formulier of rapport hebt gestart. De routine die het laatst is aangeroepen, wordt het eerst weergegeven, gevolgd door de aanroepende routine enzovoort, totdat u terug bent bij de eerste methode of procedure. Deze opdracht is alleen beschikbaar als de uitvoering wordt stopgezet op een afbreekpunt.
- *Afbreekpunt instellen* stelt afbreekpunten in een methode in om de uitvoering te stoppen op opgegeven regels. U kunt zoveel afbreekpunten instellen als uw systeemgeheugen toestaat.

***Opmerking*** Alle afbreekpunten worden verwijderd als u uw code verandert.

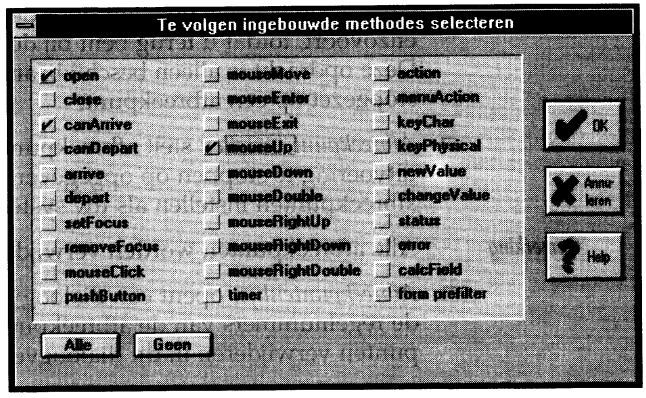
- *Afbreekpuntenlijst* opent een dialoogvenster met een overzicht van de regelnummers van de afbreekpunten. U kunt ook afbreekpunten verwijderen in dit dialoogvenster.
- *Uitvoering volgen* opent een venster en toont de ObjectPAL-coderegel die wordt uitgevoerd. De instelling van deze optie wordt opgeslagen met uw formulier. U hoeft de optie dus niet telkens als u de uitvoering wilt kunnen volgen, opnieuw te selecteren. Als deze optie niet is geselecteerd, verloopt de uitvoering normaal.

  De ObjectPAL-Tracer voert deze functie uit. Zie Hoofdstuk 13 voor een voorbeeld van het gebruik van de ObjectPAL-Tracer. ObjectPAL bevat ook procedures voor de besturing van de ObjectPAL-Tracer. Raadpleeg het onderwerp "System" in de ObjectPAL online Help.
- *Ingebouwde methodes volgen* opent een dialoogvenster met een overzicht van alle ingebouwde methodes (zie Afbeelding 10-2). Selecteer een ingebouwde methode om informatie over de methode weer te geven in het Tracer-venster, terwijl de methode wordt uitgevoerd. Deze optie werkt alleen als u aan minstens één object ObjectPAL-code hebt gekoppeld.

***Opmerking*** Als u vakken in dit dialoogvenster selecteert, wordt de Tracer niet automatisch geactiveerd. U moet ook nog 'Debug | Uitvoering volgen' (hiervoor beschreven) of 'Debug | Debug-instructie mogelijk' (hierna beschreven) kiezen en de procedure **tracerOn** gebruiken in ObjectPAL-code die aan een object is gekoppeld.

Als u een ingebouwde methode selecteert, geeft u aan dat u de uitvoering van die methode wilt volgen. Methodes die niet zijn geselecteerd, worden niet gevolgd. Een geselecteerde methode wordt altijd gevolgd, ongeacht of er ObjectPAL-code aan de methode is gekoppeld.

Afbeelding 10-2 Het dialoogvenster 'Te volgen ingebouwde methodes selecteren'




Als u een ingebouwde methode selecteert in dit dialoogvenster, geeft u aan dat u de uitvoering van deze methode wilt volgen, ongeacht of er ObjectPAL-code aan deze methode is gekoppeld. In deze afbeelding zijn de ingebouwde methodes **open**, **canArrive** en **mouseUp** geselecteerd. De uitvoering van deze methodes wordt dus gevolgd.

U kunt 'Alle' kiezen om alle ingebouwde methodes te selecteren of 'Geen' om alle methodes te deselecteren.

Als het vak met het label "form prefilter" is geselecteerd, worden methodes gevolgd tijdens de uitvoering voor het formulier en het beoogde doelobject. Anders wordt de uitvoering van methodes alleen gevolgd voor het doelobject. Zie Hoofdstuk 12 voor meer informatie over het actiemodel en de **isPreFilter**-methode.

- *Debug-instructie mogelijk* stopt de uitvoering bij Debug-instructies. Wanneer u een Debug-instructie in een methode plaatst heeft dit hetzelfde resultaat als wanneer u een afbreekpunt instelt op dezelfde regel. Het voordeel van de Debug-instructie is dat deze in de broncode wordt opgeslagen. U hoeft deze dus niet telkens opnieuw in te stellen, zoals een afbreekpunt. De instelling wordt opgeslagen met uw formulier.

  Als u deze optie selecteert, geeft Paradox bovendien gedetailleerdere foutinformatie. Deze optie kan dus nuttig zijn, zelfs als u nooit een Debug-instructie gebruikt.

  Als deze optie niet is geselecteerd, worden de Debug-instructies genegeerd. Vergeet niet deze optie te selecteren voordat u een formulier aanmaakt, anders worden de Debug-instructies uitgevoerd als een gebruiker van uw applicatie het formulier start.

- *Ctrl-Break naar debugger* onderbreekt de uitvoering en opent een Debugger-venster als u op *Ctrl-Break* drukt als deze optie is geselecteerd. Het Debugger-venster bevat de actieve methode of procedure, net als bij een afbreekpunt.

Als deze optie niet is geselecteerd, stopt u de uitvoering als u op *Ctrl-Break* drukt.

***Opmerking*** *Ctrl-Break* stopt alleen de uitvoering van ObjectPAL-methodes en -procedures. Andere bewerkingen (bijvoorbeeld queries) worden hierdoor niet beïnvloed.

- *Bron tonen* toont de code van een andere methode in een venster van de ObjectPAL-Editor. Dit is een snelle manier om een bepaalde methode van een bepaald object op te roepen. U kunt met de muis of de pijltoetsen door de elementen van dit dialoogvenster schuiven, hoewel het venster geen schuifbalken bevat.
- *Afkomst* toont de methode die het huidige afbreekpunt bevat, met de invoegpositie op de regel met het afbreekpunt. Als de uitvoering op een afbreekpunt tijdelijk is onderbroken, verschijnt er een Editor-venster met de methode die het afbreekpunt bevat.

  U kunt *Afkomst* gebruiken om snel terug te keren naar het afbreekpunt als u meerdere Editor-vensters op uw scherm hebt geopend. Deze functie is alleen beschikbaar als de uitvoering tijdelijk is onderbroken op een afbreekpunt.
- *Overheen stappen* loopt stap voor stap door een methode, waarbij procedures en eigen methodes als enkele stappen worden behandeld. Deze functie is alleen beschikbaar als de uitvoering tijdelijk is onderbroken op een afbreekpunt.
- *Stappen in* loopt stap voor stap door elke regel in een methode en door elke regel in de procedures en eigen methodes die de methode aanroept. Deze functie is alleen beschikbaar als de uitvoering op een afbreekpunt stopt.
- *Methode afbreken* stopt de uitvoering en sluit alle Debugger-vensters. Deze functie is alleen beschikbaar als de uitvoering tijdelijk is onderbroken op een afbreekpunt.
- *Starten* verlaat de Debugger, slaat de wijzigingen op en schakelt van een ontwerpvenster over naar de optie 'Gegevens tonen.' Als de uitvoering tijdelijk is onderbroken op een afbreekpunt, gaat 'Debug | Starten' door vanaf het afbreekpunt.

***Opmerking*** Als er afbreekpunten zijn ingesteld of als 'Uitvoering volgen', 'Debug-instructie mogelijk' of 'Ctrl-Break naar debugger' zijn geselecteerd, verloopt de uitvoering langzamer dan normaal.

## Debugger verlaten

Er is geen expliciete opdracht om de Debugger te verlaten. Als u 'Debug | Starten' kiest of op het onderliggende formulier klikt, keert u terug naar uw formulier. Als u de Debugger wilt verlaten, sluit u het Debugger-venster.

# Zelfstudie voor de Debugger

Deze paragraaf bevat een korte praktische zelfstudie voor de Debugger. In deze zelfstudie komen de volgende onderwerpen aan bod:

- Methodes controleren op syntaxisfouten
- Afbreekpunten toevoegen en verwijderen
- Variabelen inspecteren
- Stap voor stap delen van uw code doorlopen

## Aan de slag

U begint uw zelfstudie met een schone lei en maakt een formulier waarin u de Debugger gaat gebruiken.

1. Kies 'Bestand | Nieuw | Formulier'. Er verschijnen verscheidene dialoogvensters. Kies in het juiste dialoogvenster 'OK' om een leeg formulier te maken, dat aan geen enkele tabel is gekoppeld.
2. Inspecteer de pagina zodat het bijbehorende menu wordt geopend. Noem de pagina vervolgens *mijnPagina*.
3. Inspecteer *mijnPagina* en kies 'Methodes' om het methodevenster te openen.
4. Kies het invoervak met het label 'Nieuwe eigen methode'. Typ **eenToevoegen** en kies vervolgens 'OK'.
5. Er verschijnt een Editor-venster met de volgende standaardtekst:

```
method eenToevoegen()

endmethod
```

6. Bewerk de methode zodat deze er als volgt uitziet:

```
method eenToevoegen(var mijnGetal SmallInt) SmallInt
return mijnGetal + 1
endMethod
```

7. Sluit het Editor-venster en sla de wijzigingen op wanneer dit wordt gevraagd.
8. Gebruik het Knop-hulpmiddel om een kleine knop te plaatsen in de linkerbovenhoek van het formulier. Noem deze knop *mijnKnop*.
9. Inspecteer de knop en kies 'Methodes' om het methodevenster te openen.
10. Kies **pushButton** en kies vervolgens 'OK' om een Editor-venster te openen (of dubbelklik op **pushButton**).

11. Typ de volgende methode:

```
method pushButton(var eventInfo Event)
var
   mijnGetal, dumGetal SmallInt
endVar
mijnGetal = 5
mijnGetal = eenToevoegen(mijnGetal)
dumGetal = 123
mijnGetal.view()
endMethod
```

12. Kies 'Taal | Syntaxis controleren' om de code te controleren op syntaxisfouten en verbeter deze indien nodig.

13. Sluit het Editor-venster en sla de nieuwe **pushButton**-methode op.

## Afbreekpunten instellen

Afbreekpunten zijn nuttig als u uw code op een bepaalde plaats wilt stoppen, zodat u kunt zien wat er gebeurt. Als u bijvoorbeeld een ingewikkelde methode hebt gemaakt die niet werkt, en niet de uitvoering van alle regels wilt volgen, kunt u het best net vóór het punt waar uw methode ophoudt, een afbreekpunt instellen. Dit bespaart u typewerk en tijd.

Deze paragraaf beschrijft hoe u afbreekpunten instelt.

1. Als de syntaxis correct is, slaat u het werk op en start u het formulier.

2. Klik op de knop om de **pushButton**-methode uit te voeren. Er verschijnt een dialoogvenster met de mededeling dat de waarde van *mijnGetal* 6 is. Kies 'OK'.

3. Kies 'Formulier | Ontwerpen' om terug te keren naar een ontwerpvenster.

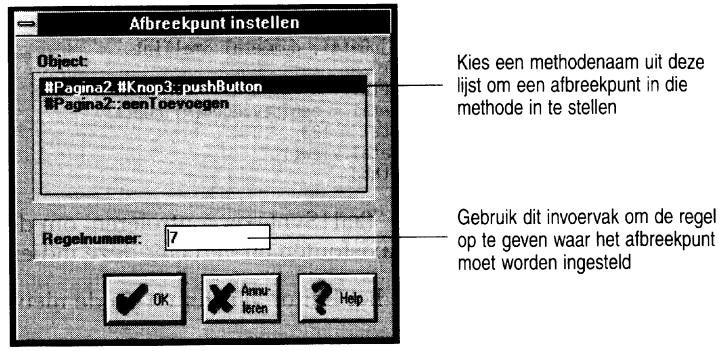
4. Geef de **pushButton**-methode weer in een Editor-venster.

5. Verplaats de invoegpositie met de muis of de pijltoetsen naar regel 7 van de methode (*dumGetal = 123*).

6. Kies 'Debug | Afbreekpunt instellen'. Er verschijnt een dialoogvenster met een overzicht van de objecten op het huidige formulier en een regelnummer. Het regelnummer is standaard de regel waarin de invoegpositie zich bevindt, maar u kunt ook rechtstreeks een regelnummer typen.

7. Kies 'OK' om een afbreekpunt in te stellen op regel 7.

Afbeelding 10-3 Een afbreekpunt instellen




8. Kies 'Debug | Starten' om het ontwerpvenster te verlaten. Klik vervolgens op de knop *mijnKnop*. De **pushButton**-methode wordt uitgevoerd tot aan de regel waar u het afbreekpunt hebt ingesteld. Daarna verschijnt de methode in een Editor-venster.

9. Kies 'Debug | Starten' (of kies 'Start dit formulier' op de TurboBalk) om de resterende regels van de methode uit te voeren. De uitvoering wordt hervat vanaf de regel die het afbreekpunt bevat.

## Variabele inspecteren

Als de Debugger een afbreekpunt tegenkomt, kunt u variabelen inspecteren om de waarden te bekijken. Dit is handig als u fouten zoekt in code waarvan de syntaxis correct is, maar die toch niet goed werkt.

Als u het nieuwe formulier (en het afbreekpunt) wilt gebruiken om een variabele te inspecteren, gaat u als volgt te werk:

1. Klik op de knop *mijnKnop* om de **pushButton**-methode vanaf het begin uit te voeren. De uitvoering wordt tijdelijk onderbroken als het afbreekpunt wordt bereikt.

2. Plaats de invoegpositie in het woord *mijnGetal* (elke *mijnGetal* in de methode is goed).

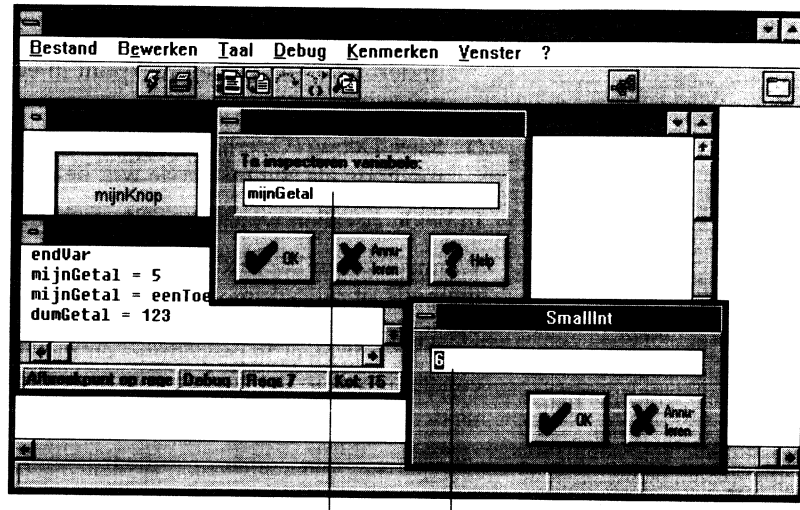
3. Kies 'Debug | Inspecteren'. Er verschijnt een dialoogvenster met de naam van een variabele die u kunt inspecteren. Dit dialoogvenster toont standaard het woord waarin de invoegpositie zich bevindt (ongeacht of dit een variabele is), in dit geval *mijnGetal*. U kunt ook met de muis slepen om een element te selecteren dat u wilt inspecteren.

4. Kies 'OK'. Er verschijnt een nieuw dialoogvenster dat de waarde van *mijnGetal* toont (6 in dit geval). Deze waarde is gemarkeerd, wat betekent dat u de waarde van *mijnGetal* kunt veranderen door

een nieuwe waarde te typen in het tekstvak. Kies in dit geval 'OK' om de getoonde waarde zo te laten.

5. Kies opnieuw 'Debug | Inspecteren'. Typ **dumGetal** in het tekstvak en kies 'OK'. Het dialoogvenster geeft **N/A** weer, omdat er op deze plaats in de code nog geen waarde is toegewezen aan de variabele *dumGetal*. (Een methode wordt uitgevoerd *totdat* er een afbreekpunt wordt bereikt. De regel met het afbreekpunt wordt niet uitgevoerd.)

Afbeelding 10-4 Een variabele inspecteren




Geef hier aan welke variabele u wilt inspecteren. Kies vervolgens 'OK'

Er verschijnt een dialoogvenster met de waarde van de opgegeven variabele

## Afbreekpunten verwijderen

U kunt afbreekpunten het best verwijderen zodra u ze niet meer nodig hebt. Als u het afbreekpunt wilt verwijderen dat u zojuist hebt toegevoegd, gaat u als volgt te werk:

1. Kies 'Debug | Afbreekpuntenlijst' om een dialoogvenster te openen met alle afbreekpunten in alle methodes op het formulier. (Dit voorbeeld toont één methode en één afbreekpunt.)
2. Kies het afbreekpunt en kies vervolgens 'Verwijderen' om het afbreekpunt te verwijderen.

***Opmerking*** Afbreekpunten worden verwijderd als u uw code verandert.

### Stappen in en Overheen stappen

U kunt ook afbreekpunten gebruiken om stap voor stap door de coderegels te lopen om te zien wat er gebeurt als de methode wordt uitgevoerd. Dit kan u helpen erachter te komen waarom een variabele een verkeerde waarde krijgt.

De **pushButton**-methode in het voorbeeld roept bijvoorbeeld de eigen methode **eenToevoegen** aan. Als u fouten zoekt in de **pushButton**-methode, kunt u kiezen of u de uitvoering van elke opdracht in **eenToevoegen** wilt volgen of **eenToevoegen** als een afzonderlijke instructie wilt behandelen en alleen de uitvoering van opdrachten in het hoofddeel van de **pushButton**-methode wilt volgen. Als u in of over procedures wilt stappen, gaat u als volgt te werk:

- Kies 'Debug | Stappen in' als u de uitvoering tijdelijk wilt onderbreken op een afbreekpunt in een eigen methode of procedure.
- Kies 'Debug | Overheen stappen' als u de procedure of eigen methode als één instructie wilt behandelen.

In beide gevallen wordt dezelfde code uitgevoerd.

Wanneer zou u bepaalde delen van de code willen overslaan? Als een methode of procedure goed werkt, heeft het geen zin om deze telkens opnieuw te onderzoeken als een methode deze aanroept. Als u over goed gecontroleerde procedures en methodes heenstapt, kunt u zich concentreren op het zoeken van fouten in de methode waar het om gaat.

## Snelle manieren

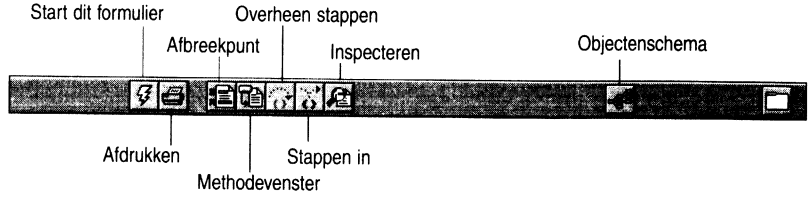
De ObjectPAL-Editor bevat TurboBalk-knoppen en ondersteunt de rechtermuisknop voor veel voorkomende handelingen.

### TurboBalk-knoppen

- *Formulier starten* heeft dezelfde werking als de menu-optie 'Debug | Starten'.
- *Afbreekpunt instellen* heeft dezelfde werking als de optie 'Debug | Afbreekpunt instellen'.
- *Methodevenster* heeft dezelfde werking als de optie 'Taal | Methodes'.
- *Overheen stappen* heeft dezelfde werking als de optie 'Debug | Overheen stappen'.
- *Stappen in* heeft dezelfde werking als de optie 'Debug | Stappen in'.
- *Inspecteren* heeft dezelfde werking als de optie 'Debug | Inspecteren'.

❒ *Objectenschema* heeft dezelfde werking als de optie 'Taal | Objectenschema'.

Afbeelding 10-5 De TurboBalk van de Debugger




## Rechtermuisknop

Als u fouten opspoort in een methode, klikt u rechts om een pop-up menu te openen. Dit menu bevat dezelfde opties als het menu 'Debug'.

Als u een methode bewerkt, klikt u rechts om een pop-up menu te openen met dezelfde opties als het menu 'Taal'.

HOOFDSTUK 11

# Taalstructuur

In dit hoofdstuk wordt de algemene structuur en de syntaxis van ObjectPAL-methodes besproken, inclusief de regels voor de benoeming en het gebruik van variabelen en constanten. De volgende onderwerpen komen aan de orde:

- De aard van ObjectPAL
- Puntnotatie
- Structuur
- Controlestructuren
- Gegevenstypes
- Uitdrukkingen
- Benoemingsregels
- ObjectPAL-termen
- Insluitende objecten
- Het gebruik van variabelen
- De definitie van constanten
- Het doorgeven van argumenten aan methodes en procedures

## De aard van ObjectPAL

Ervaren programmeurs zullen zich afvragen wat het verschil is tussen ObjectPAL-applicaties en traditionele procedurele programma's. De belangrijkste verschillen betreffen de context van het programma en de volgorde van de uitvoering. De context van een traditioneel programma bestaat uit één bronbestand en de volgorde van uitvoering is lineair. In ObjectPAL wordt de context bepaald door objecten en worden methodes uitgevoerd in reactie op acties.

Voor wat betreft de structuur en de syntaxis, lijken ObjectPAL-methodes op traditionele programma's. Bijvoorbeeld:

- Methodes zijn gestructureerd. Naast de uitvoering van het begin tot het einde kunt u een geordende uitvoeringsstructuur definiëren, omdat ObjectPAL controlestructuren en -lussen ondersteunt, zoals **while...endWhile**, **if...then...else** en **switch...case...endSwitch**.
- Methodes kunnen parameters hebben (ook wel argumenten genoemd). In de volgende instructie is DataNextRecord bijvoorbeeld de parameter, **action** de methode en *orderTabel* het object:

```
orderTabel.action(DataNextRecord)
```

- Net als in Pascal en C kunt u procedures definiëren die één of meer taken uitvoeren. Procedures kunnen argumenten ontvangen van en resultaten teruggeven aan de methode die de procedures aanroept.
- Net als in C kunt u vrijelijk witruimte gebruiken (tabs, spaties en witregels). U kunt ondergeschikte methoderegels desgewenst laten inspringen, één of meer instructies op een regel plaatsen en aan elke methoderegel commentaar toevoegen. Witruimte heeft geen invloed op de uitvoering van instructies.

# Puntnotatie

ObjectPAL gebruikt een puntnotatie om elementen in een instructie van elkaar te scheiden. De volgende instructie wordt gelezen als "Kader één punt kleur is gelijk aan blauw".

```
kaderEen.color = Blue
```

Deze instructie zegt dat u wilt werken met het object dat *kaderEen* heet en dat u het kenmerk 'Color' van *kaderEen* op blauw wilt instellen. De punten scheiden de objectnaam en de kenmerknaam. De volgende instructie wordt gelezen als "Mijn formulier punt titel instellen op 'Bestelformulier'".

```
mijnForm.setTitle("Bestelformulier")
```

De punt scheidt de variabele *mijnForm* en de methodenaam **setTitle**.

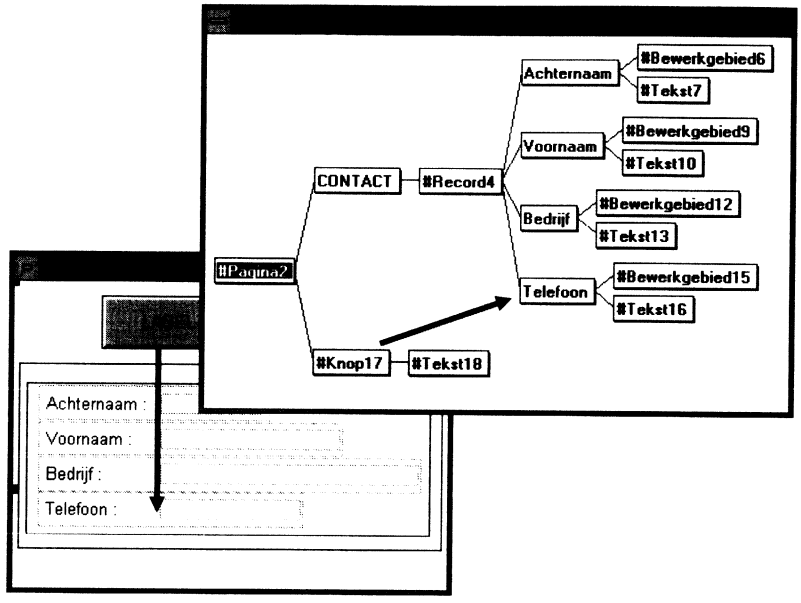
***Belangrijk*** In de tekst wordt (anders dan in de codevoorbeelden) in dit handboek alleen naar methodes en procedures verwezen met de naam en niet met de volledige syntaxis. De vorige alinea verwijst bijvoorbeeld naar de **setTitle**-methode die is gedefinieerd voor het type Form. De volledige syntaxis van de **setTitle**-methode is

**setTitle** ( const ***nwTitel*** String )

Raadpleeg de ObjectPAL online Help voor de volledige syntaxis van de ObjectPAL-methodes en procedures.

Zie Afbeelding 11-1 voor een complexer voorbeeld van de puntnotatie. In deze afbeelding wordt een formulier getoond dat een knop en een tabelframe bevat, die zijn verbonden met de tabel *Contact*. De afbeelding toont ook het Objectenschema dat een diagram toont van de relaties tussen de objecten.

Afbeelding 11-1 Een object met een unieke naam adresseren




Op dit formulier bevindt zich slechts één object met de naam *Telefoon*. Hierdoor kan de knop de volgende instructie gebruiken om de waarde van *Telefoon* in te stellen:

Telefoon.value = "020-5550147"

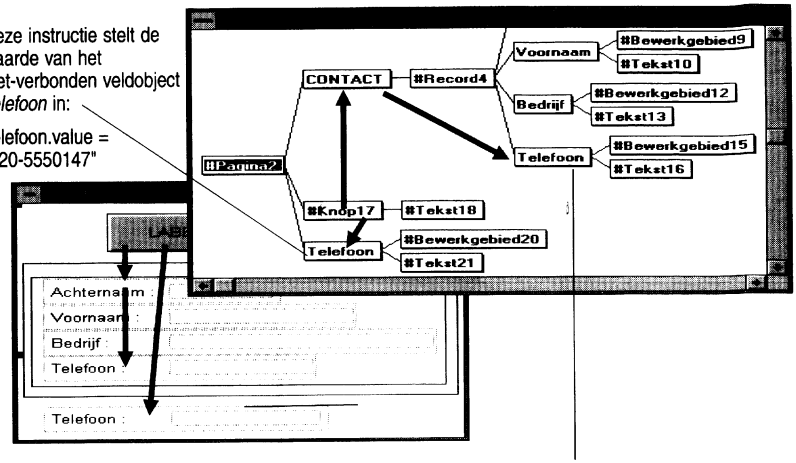
Code die is gekoppeld aan de **pushButton**-methode van de knop, kan de volgende instructie gebruiken om de waarde in te stellen van het veldobject *Telefoon* in de tabel *Contact*:

```
method pushButton(var eventinfo Event)
   Telefoon.value = "020-5550147"  ; alleen in bewerkmodus
endmethod
```

Als een object een unieke naam heeft, kunt u dat object direct adresseren, zoals in het vorige voorbeeld. Als een of meer objecten op een formulier dezelfde naam hebben, moet u de puntnotatie gebruiken om aan te geven met welk object u wilt werken. In Afbeelding 11-2 ziet u bijvoorbeeld een formulier met de knop en het tabelframe uit het vorige voorbeeld. Er is nog een veldobject met de naam *Telefoon* toegevoegd. Het Objectenschema toont de relaties tussen de objecten.

Afbeelding 11-2 Een object adresseren waarvan de naam niet uniek is

Dit formulier bevat twee objecten met de naam *Telefoon*. Gebruik puntnotatie om aan te geven met welke 'Telefoon' u wilt werken.




Deze instructie stelt de waarde in van het veldobject *Telefoon*, dat zich in het tabelframe CONTACT bevindt: CONTACT.Telefoon.value = "020-5550147"

Als u nu de waarde van het veldobject *Telefoon* in de tabel *Contact* op dezelfde manier wilt instellen als in het vorige voorbeeld, moet u de puntnotatie gebruiken:

```
method pushButton(var eventinfo Event)
   Contact.Telefoon.value = "020-5550147"
endmethod
```

In dit voorbeeld stelt de volgende instructie de waarde in van het object *Telefoon*, dat zich in het object *Contact* bevindt:

```
Contact.Telefoon.value = "020-5550147"
```

Als u de waarde van het andere *Telefoon*-object, wilt instellen, luidt de code van de **pushButton**-methode als volgt:

```
method pushButton(var eventinfo Event)
   Telefoon.value = "020-5550147"
endmethod
```

***Belangrijk*** De puntnotatie zorgt voor duidelijkheid en eenheid, maar is niet vereist. Als alternatieve syntaxis kunt u de methode aanroepen en de objectnaam als het eerste argument gebruiken en andere argumenten, indien nodig, naar rechts verplaatsen. De volgende instructies zijn bijvoorbeeld aan elkaar gelijk:

```
mijnCosinus = deHoek.cos() ; puntnotatie, geen argumenten

mijnCosinus = cos(deHoek)  ; alternatieve syntaxis, objectnaam is enige argument
```

De volgende instructies zijn eveneens gelijk:

```
mijnTekst.open("tekst.txt", "r")   ; puntnotatie, twee argumenten
open(mijnTekst, "tekst.txt", "r")  ; alternatieve syntaxis, drie argumenten
```

# Structuur

ObjectPAL-methodes hebben een vrije structuur. Enkele uitzonderingen daargelaten, kunt u zoveel instructies op één regel plaatsen als u wilt, op voorwaarde dat u de instructies door tenminste één spatie scheidt.

## Regels splitsen

U kunt een instructie in twee of meer regels verdelen, als u de splitsing maar niet aanbrengt in een sleutelwoord, een naam of een waarde (zoals een getal of een tekenreeks). Elke regel in een ObjectPAL-methode kan maximaal 132 tekens bevatten. De volgende twee voorbeelden worden op dezelfde manier uitgevoerd:

```
if tc.locate("KlantNaam", "Nico Steen") then
   tc.ProduktNaam = "Paradox"
else
   tc.ProduktNaam = "dBASE"
endIf

if
tc.locate("KlantNaam", "Nico Steen")
then
tc.ProduktNaam =
"Paradox" else
tc.ProduktNaam =
"DBASE"
endIf
```

## Hoofdletters

*Geen onderscheid tussen hoofd- en kleine letters*

U kunt hoofdletters of kleine letters gebruiken of een combinatie ervan. ObjectPAL herkent bijvoorbeeld setFieldValue en SETFIELDVALUE en zelfs SeTfIElDvAlUe als hetzelfde woord (maar het is minder verwarrend als u hoofdletters en kleine letters consequent gebruikt). Hetzelfde geldt voor de namen van tabellen, velden, variabelen, arrays en procedures.

De enige plaats waar het onderscheid wel van belang is in uw methodes, is in reekswaarden (alfanumeriek). De reeks "abc" is bijvoorbeeld niet gelijk aan "ABC".

***Opmerking***

U kunt **ignoreCaseInStringCompares**, gedefinieerd voor het String-type, gebruiken als u wilt dat het verschil tussen hoofdletters en kleine letters wordt genegeerd bij de vergelijking van reeksen.

## Commentaar en witregels

ObjectPAL kent twee manieren om commentaar in code te plaatsen:

- Puntkomma's
- Accolades

### *Puntkomma*

Met uitzondering van reekswaarden wordt alles wat in een methoderegel na een puntkomma (;) komt, als commentaar

beschouwd en door ObjectPAL genegeerd. Als u een regel met een puntkomma begint, wordt de hele regel als commentaar behandeld. Als commentaar uit meerdere regels bestaat, moet elke regel met een puntkomma beginnen, zoals in het volgende voorbeeld:

```
method pushButton (var eventInfo Event)

; eerst variabelen declareren
var mijnGetal SmallInt endVar

for mijnGetal from 1 to 5
   mijnGetal = mijnGetal + 1 ; dit is een tellus
endFor

; dit commentaar neemt
; twee regels in beslag

endMethod
```

Gebruik commentaar om te beschrijven wat er in een methode gebeurt. Bijvoorbeeld om cryptische berichten, variabelen en procedures te verklaren of andere informatie te geven die nuttig kan zijn voor iemand die de methode leest of bewerkt of om u zelf eraan te herinneren wat u hebt gedaan. Als u commentaar aan een methode toevoegt, vertraagt dit de uitvoering niet.

Witregels en witruimte worden genegeerd. U kunt witregels gebruiken om commentaar te onderscheiden en uw methodes leesbaarder te maken. U hoeft voor een witregel geen puntkomma te plaatsen.

## Accolades

U kunt accolades gebruiken om van grote blokken code commentaar te maken. U gebruikt een linkse accolade ({) om een commentaarblok te beginnen en een rechtse accolade (}) om het blok te beëindigen. De volgende twee voorbeelden maken van dezelfde regels commentaar.

*Voorbeeld 1*

```
var
   i SmallInt
endVar
{
for i from 1 to 10
   i = i + 1
endFor
}
msgInfo("Status", "U bent hier.")
```

*Voorbeeld 2*

```
var
   i SmallInt
endVar

; for i from 1 to 10
;   i = i + 1
; endFor

msgInfo("Status", "U bent hier.")
```

### Reeksen tussen aanhalingstekens

Een reeks tussen aanhalingstekens die in een methode meer dan één regel in beslag neemt, neemt hetzelfde aantal regels in beslag als deze wordt weergegeven. De volgende instructie:

```
msgInfo("2-regel reeks", "Deze reeks
neemt twee regels in beslag.")
```

geeft bijvoorbeeld een dialoogvenster weer met het volgende bericht:

```
Deze reeks
neemt twee regels in beslag.
```

# Controlestructuren

ObjectPAL voert instructies uit in de volgorde waarin deze in een methode verschijnen. ObjectPAL kent echter ook *controlestructuren* waarmee u het *verloop*, de volgorde waarin instructies worden uitgevoerd, kunt veranderen.

De elementaire taalonderdelen van ObjectPAL bevatten controlestructuren die de volgende taken uitvoeren:

- *Sprongen* naar een bepaalde instructie of groep instructies uit de instructies die u opgeeft. De keuze wordt gebaseerd op een voorwaarde die wordt bekeken als de methode wordt uitgevoerd. De elementaire taalonderdelen voor sprongen zijn **if**, **iif** en **switch**.
- *Lussen* herhalen een instructie of een aantal instructies totdat aan een bepaalde voorwaarde is voldaan. De elementaire taalonderdelen voor lussen zijn **for**, **forEach**, **lus**, **scan**, en **while**.
- Lussen kunnen *beëindigd* worden. De elementaire taalonderdelen voor de beëindiging van een lus zijn **quitLoop** en **return**.

Raadpleeg de ObjectPAL online Help voor meer informatie en voorbeelden.

# Gegevenstypes

In Tabel 11-1 wordt een overzicht gegeven van de gegevenstypes van ObjectPAL. Deze gegevenstypes en complexere types, worden in Hoofdstuk 15 beschreven.

Tabel 11-1 ObjectPAL-gegevenstypes

| Type | Bevat |
|---|---|
| Alphanumeric | Maximaal 32.767 tekens voor een String-variabele, maximaal 255 tekens voor een reeks tussen aanhalingstekens |
| Currency | Geldwaarden van $\pm 3{,}4 * 10^{-4930}$ tot $\pm 1{,}1 * 10^{4930}$, tot zes decimalen exact |
| Date | Datums van 1 januari 100 tot 31 december 9999 |
| DateTime | Een tijd en een datum |
| Graphic | Bitmap-beelden |
| Logical | True of False |
| LongInt | Integerwaarden van -2.147.483.648 tot 2.147.483.647 (vier bytes) |
| Memo | Maximaal 512 MB in Paradox-tabellen |
| Number | Waarden voor getallen met zwevend decimaalteken van $\pm 3{,}4 * 10^{-4930}$ tot $\pm 1{,}1 * 10^{4930}$ |
| SmallInt | Integerwaarden van -32.768 tot 32.767 (twee bytes)* |
| Time | Klokgegevens in de vorm uur-minuut-seconde-milliseconde |

* -32.768 kan niet in een Paradox-tabel worden opgeslagen, omdat -32.768 voor Paradox leeg is.

Tabel 11-2 laat zien hoe u waarden van verschillende types kunt combineren en converteren.

***Belangrijk*** U kunt niet alle types combineren. U kunt bijvoorbeeld een String niet bij een Number optellen. Binary-, Graphic- en OLE-types kunnen helemaal niet worden gecombineerd en zijn daarom niet in de tabel opgenomen.

Tabel 11-2 Gegevenstypes combineren

| | M | A | E | T | D | C | N | L | S | & |
|---|---|---|---|---|---|---|---|---|---|---|
| M (Memo) | M | * | * | * | * | * | * | * | * | * |
| A (String) | * | A | * | * | * | * | * | * | * | * |
| E (DateTime) | * | * | E | E | E | * | E | * | E | * |
| T (Time) | * | * | E | T | T | * | T | * | * | * |
| D (Date) | * | * | E | T | D | * | D | D | D | * |
| C (Currency) | * | * | * | * | * | C | C | C | C | * |
| N (Number) | * | * | E | T | D | C | N | N | N | * |
| L (LongInt) | * | * | * | * | D | C | N | L | L | * |
| S (SmallInt) | * | * | E | * | D | C | N | L | S | * |
| & (Logical) | * | * | * | * | * | * | * | * | * | & |

* niet toegestaan

Binary-, Graphic-, en OLE-types kunnen niet worden gecombineerd, en zijn daarom uit de tabel weggelaten.

## Gegevenstypes converteren

U kunt een gegevenstype expliciet naar een ander gegevenstype converteren met behulp van de juiste omzettingsprocedure, zoals in het volgende voorbeeld wordt getoond. De regels voor omzettingen (casting) zijn minder beperkend dan de regels die ObjectPAL hanteert voor automatische conversie. U kunt een waarde naar een minder exact gegevenstype converteren, maar het resultaat kan een ingekorte versie zijn van de oorspronkelijke waarde.

```
var
   mijnGetal Number ; declareert mijnGetal als een Number
   hetResultaat   String
endVar

message(1 + 2)                    ; Geeft 3 weer
sleep(1500)

message(1.0 + 2)                  ; Geeft 3,00 weer
sleep(1500)

mijnGetal = 3.65                  ; Wijst de waarde 3,65 toe aan mijnGetal
mijnGetal = SmallInt(3.65)        ; Verandert Number 3,65 in SmallInt, geeft
                                  ; 3,00 terug
mijnGetal = SmallInt("12.67")     ; Verandert String "12.67" in SmallInt,
                                  ; geeft 12,00 terug

hetResultaat = "abc" + String(mijnGetal)    ; hetResultaat = "abc12"
msgInfo("Het resultaat is", hetResultaat)
```

Kijk ook naar het volgende voorbeeld. Omdat de variabelen *a* en *b* worden gedefinieerd als SmallInt, berekent ObjectPAL het antwoord ook als een SmallInt:

```
var
   a, b SmallInt
endVar
```

```
a = 1
b = 2
message(a/b)  ; geeft 0 weer
              ; a/b = 1/2 = 0,5, dus een integerwaarde van 0
```

Als u het antwoord als een decimale waarde wilt berekenen, zet u *a* en *b* om naar Numbers:

```
var
   a, b SmallInt
endVar
a = 1
b = 2
message(Number(a)/Number(b))  ; geeft 0,50 weer
```

De volgende instructie geeft "0,00" weer, omdat de omzettingsprocedure wordt aangeroepen *nadat* het antwoord is berekend met SmallInt-waarden.

```
message(Number(1/2)) ; geeft 0,00 weer
```

Een andere benadering is vanaf het begin te werken met Number-waarden. De volgende instructies geven bijvoorbeeld alle "0,50" weer:

```
message(1/2.0) ; al deze instructies geven 0,50 weer
message(1.0/2)
message(1.0/2.0)
```

# Uitdrukkingen

Een uitdrukking is één waarde of een verzameling van een of meer elementen die resulteren in één waarde.

Hieronder volgen enkele voorbeelden van uitdrukkingen:

```
5                              ; het getal 5
"hallo"                        ; de reeks hallo tussen aanhalingstekens
"hallo" + " wereld"            ; telt twee reeksen op
mijnTabel.mijnVeld.value + 5   ; telt 5 op bij de waarde van mijnVeld
mijnTabel.cSum("Bedrag")       ; berekent de som van de waarden in het veld
                               ; 'Bedrag'
```

## Operatoren

U kunt operatoren in uitdrukkingen gebruiken om waarden en gegevenselementen te combineren en te manipuleren. In Tabel 11-3 wordt een overzicht gegeven van de operatoren die in ObjectPAL beschikbaar zijn. Raadpleeg de paragrafen "Array," "DynArray," "Point," en "Record" in Hoofdstuk 15 voor meer informatie over het gebruik van operatoren bij specifieke gecompliceerde gegevenstypes.

***Opmerking*** Het =-symbool dient ook als een toewijzingsoperator. Raadpleeg de ObjectPAL online Help voor meer informatie en voorbeelden.

Tabel 11-3 ObjectPAL-operatoren voor gegevenstypes

| Gegevenstype/-categorie | Operator | Functie |
|---|---|---|
| Alphanumeric (A) | | |
| | + | Aaneenschakelen |
| Numeric (N,$,S) | | |
| | + | Optellen |
| | - | Aftrekken of teken omkeren |
| | * | Vermenigvuldigen |
| | / | Delen |
| Date (D), Time (T), en DateTime (E) | | |
| | + | Optellen |
| | - | Aftrekken of verschil in dagen |
| Vergelijking | | |
| | = | Gelijk aan |
| | <> | Niet gelijk aan |
| | < | Kleiner dan |
| | <= | Kleiner dan of gelijk aan |
| | > | Groter dan |
| | >= | Groter dan of gelijk aan |
| Logical | | |
| | AND | Logische AND |
| | OR | Logische OR |
| | NOT | Logische NOT |

Ga er bij de voorbeelden in deze paragraaf vanuit dat de volgende variabelen zijn toegewezen:

```
Leeftijd = 50.2
Nieuwjaar = Date("1/1/1993")
Middag = DateTime("12:00:00 am 12-12-12")
Dodelijk = "belasting"
```

### De operator +

De werking van de operator (+) is afhankelijk van het type uitdrukking waarop de operator werkt:

- *Number* + *Number* telt de twee getallen op.

```
message(45 + 50.2 + 5)          ; 100,20 (optelling)
message(45 + Leeftijd + 5)      ; 100,20 (optelling)
```

- *String* + *String* combineert de reeksen (wordt *aaneenschakeling* genoemd); reeksaaneenschakeling is geldig voor alfanumerieke velden en memovelden.

```
message("Inkomsten" + "belasting") ; "Inkomstenbelasting" (aaneenschakeling)
message("Inkomsten" + Dodelijk)    ; "Inkomstenbelasting" (aaneenschakeling)
```

- *Date* + *Number* (of *Number* + *Date*) telt een aantal dagen bij de datum op (wordt *datumoptelling* genoemd).

```
message(7 + Date("12/17/91"))       ; 24-12-91 (datumoptelling)
message(Nieuwjaar + 1)              ; 2-01-93 (datumoptelling)
```

- *Time* + *Number* (of *Number* + *Time*) telt een aantal milliseconden op bij een tijdwaarde.

```
message(Time("1:00:00 am") + 10000) ; 01:00:10
```

- *DateTime* + *Date* telt een datumwaarde op bij een DateTime-waarde. Het resultaat is een DateTime-waarde.

```
message(Middag + Date("01/01/01")) ; 12:00:00, 12-12-3812
```

- *DateTime* + *Time* telt een tijdwaarde op bij een DateTime-waarde. Het resultaat is een DateTime-waarde.

```
message(Middag + Time("01:01:01 PM") ; 01:01:01, 12-12-12
```

- *DateTime* + *Number* (of *Number* + *DateTime*) telt een aantal milliseconden op bij een DateTime-waarde.

```
message(Middag + 5000) ; 12:00:05, 12-12-12
```

Er zijn geen andere combinaties van verschillende gegevenstypes toegestaan met de operator (+).

### De operator -

Net als de operator (+), kan de operator (-) op verschillende manieren worden gebruikt:

- *Number* - *Number* trekt het tweede getal van het eerste getal af (wordt *aftrekking* genoemd).

```
message(76.5 - Leeftijd)          ; 26,3 (aftrekking)
```

- *-Number* keert het teken van het getal om (wordt *monadische omkering* genoemd).

```
message(-Leeftijd)                ; -50,2 (omkering)
```

- *Date* - *Number* trekt een aantal dagen van een datum af (wordt *datumaftrekking* genoemd).

```
message(Nieuwjaar - 3)            ; 28-Dec-1992 (datumaftrekking)
```

- *Date* - *Date* geeft een datumwaarde terug die staat voor het aantal dagen tussen twee datums (wordt aangeduid met *verschil in dagen*). Als u het verschil in dagen in de vorm van een getal wilt uitdrukken, zet u het resultaat om naar een Number-waarde, zoals u in het volgende voorbeeld ziet:

```
var
   d1, d2 Date
   dagenVerschil Number
endVar
```

```
d1 = Date("12/21/95")
d2 = Date("12/11/95")
dagenVerschil = Number(d1 - d2)
message(dagenVerschil) ; geeft 10 weer

message(Number(Today() - Nieuwjaar))    ; geeft aantal dagen weer sinds
                                        ; 1 januari 1993
```

Er zijn geen andere combinaties van argumenten toegestaan met de operator -. U kunt - met name niet gebruiken in combinatie met reeksargumenten. (U kunt echter de String-methode **substr** gebruiken om subreeksen uit reeksen te halen.)

### De operatoren * en /

U kunt de vermenigvuldigingsoperator (*) en de delingsoperator ( / ) alleen in combinatie met numerieke argumenten gebruiken. Deling door nul leidt tot een runtime fout.

### Vergelijkingsoperatoren

U kunt de vergelijkingsoperatoren uit Tabel 11-4 gebruiken om de waarden van gegevenstypes, inclusief logische waarden, te vergelijken. Het resultaat is altijd een logische waarde, True of False. De werking van operatoren zoals (<) en (>) is afhankelijk van de types van de argumenten die worden vergeleken.

Tabel 11-4 Vergelijkingsoperatoren

| Gegevenstype | Basis | Uitgangspunt |
|---|---|---|
| Numeric | Numerieke volgorde | Lager < Hoger |
| Alphanumeric | Alfanumerieke volgorde | Lager < Hoger |
| Datum | Chronologische volgorde | Vroeger < Later |
| Logisch | | False < True |

Alfanumerieke vergelijkingen zijn afhankelijk van de sorteervolgorde. De ASCII-sorteervolgorde is A < Z < a < z, maar dit geldt niet voor systemen die een woordenboekvolgorde ondersteunen (dat wil zeggen, tekens worden gesorteerd in alfabetische volgorde ongeacht hoofdletters en kleine letters). Lege waarden worden als kleiner beschouwd dan alle andere waarden van hetzelfde gegevenstype.

U kunt alfanumerieke velden (A) met memovelden (M) vergelijken. Als u waarden van verschillende types vergelijkt, geeft (<>) een True-waarde terug, terwijl alle andere operatoren False teruggeven, zoals in het volgende voorbeeld:

```
message(5 + 1 < 6 * 2)              ; True (zie ook "Verwerkingsvolgorde")
message(Leeftijd > 21)              ; True
message(Date(5/5/1945) = Nieuwjaar) ; False, omdat de datums niet hetzelfde
                                    ; zijn
message("Abc" = "ABC")              ; False, omdat de reeksen niet hetzelfde
                                    ; zijn
message("Abc" > "ABC")              ; True in ASCII-volgorde, False in andere
                                    ; volgorden
message(Leeftijd > Leeftijd + 5)    ; False
```

```
message(3 = "ABC")                    ; False, omdat de types niet overeenkomen
message(3 <> "ABC")                   ; True, omdat de types niet overeenkomen
```

**Opmerking** Het teken (=) wordt als een vergelijkingsoperator binnen uitdrukkingen gebruikt *en* als onderdeel van de toewijzingsinstructie. Als er links van het teken (=) alleen een naam van een variabele staat, krijgt de variabele de waarde van de uitdrukking die rechts staat; anders worden de twee uitdrukkingen vergeleken.

```
a = 3       ; wijst de waarde 3 aan a toe
x = 4 = a   ; de eerste = is een toewijzing, de tweede een vergelijking
```

De vorige instructie wijst aan de variabele *x* de waarde van de uitdrukking 4 = a (False) toe. Plaats uitdrukkingen tussen haakjes als er kans op verwarring bestaat:

```
x = (4 = a)
```

## Logische operatoren (AND, OR, NOT)

U kunt uitdrukkingen met logische waarden combineren met behulp van de logische operatoren uit Tabel 11-3:

- x AND *y* geeft True terug als beide argumenten (*x* en *y*) waar zijn.
- x OR *y* geeft True terug als een van de argumenten waar is (of beide).
- NOT *x* geeft True terug als het argument onwaar is.

In ObjectPAL worden de uitdrukkingen aan beide zijden van een AND-instructie of een OR-instructie berekend, dit in tegenstelling tot veel andere programmeertalen.

Hieronder volgen enkele voorbeelden:

```
message(3 = 4 AND 7 = 8)          ; False, omdat geen van de argumenten waar is
message(3 = 4 OR 7 = 8)           ; False, omdat beide argumenten onwaar zijn
message(3 = 3 OR 7 = 8)           ; True, één argument is waar
message(NOT (3 = 4 AND 7 = 8))    ; True, omdat het argument onwaar is
message(NOT False)                ; True
message(3 <> 4)                   ; True
```

**Opmerking** Er bestaat een verschil tussen de vergelijkingsoperator (<>) (niet gelijk aan) en de logische operator NOT. De operator (<>) vergelijkt twee gegevenswaarden, maar de NOT-operator keert een logische waarde om.

ObjectPAL kent ook methodes voor de uitvoering van bitsgewijze handelingen. De types LongInt en SmallInt bevatten de methodes **bitAND**, **bitOR** en **bitXOR**. Raadpleeg de ObjectPAL online Help voor meer informatie.

**Opmerking** Voor ObjectPAL zijn de volgende instructies gelijk:

```
if x = True then y endIf

if x then y endIf
```

Ook de volgende twee instructies zijn gelijk:

```
if x = False then y endIf

if not x then y endIf
```

## Verwerkingsvolgorde

In uitdrukkingen met meer dan één operator, worden de handelingen berekend in de volgorde van prioriteit die in Tabel 11-5 wordt getoond.

***Belangrijk*** ObjectPAL gebruikt *niet* de korte Booleaanse evaluatie. Beide zijden van een logische (Booleaanse) uitdrukking worden berekend voordat de uitvoering verder gaat.

Uitdrukkingen tussen haakjes worden eerst verwerkt en de binnenste haakjesniveaus worden vóór de buitenste niveaus verwerkt. Als twee of meer operatoren van gelijke prioriteit zich in één uitdrukking bevinden, worden deze van links naar rechts verwerkt, zoals in de volgende code:

```
3 + 4 * 5           ; geeft 23 terug (* heeft prioriteit ten opzichte van +)
(3 + 4) * 5         ; geeft 35 terug (haakjes eerst)
((3 + 4) * 5) / 2   ; geeft 17,5 terug (binnenste haakjes eerst)
```

Tabel 11-5 Prioriteit van operatoren

| Prioriteit | Operatoren |
|---|---|
| 1 | () |
| 2 | * / |
| 3 | + - |
| 4 | = <> < <= > >= |
| 5 | NOT |
| 6 | AND |
| 7 | OR |

Tabel 11-6 geeft een overzicht van andere symbolen die in ObjectPAL-uitdrukkingen worden gebruikt.

Tabel 11-6 Symbolen die in ObjectPAL worden gebruikt

| Teken | Functie |
|---|---|
| ~ | Tilde (query-variabele) |
| _ | Voorlooponderstrepingsteken (query-voorbeeldelement) |
| ; | Puntkomma (commentaar) |
| { | Linkse accolade (begin commentaarblok) |
| } | Rechtse accolade (einde commentaarblok) |

In Tabel 11-7 wordt een overzicht gegeven van de symbolen voor het zoeken naar tekenreeksen.

Tabel 11-7 Zoeksymbolen die bij tekenreeksen worden gebruikt

| Symbool | Functie |
|---|---|
| .. | Stemt overeen met alle tekens |
| @ | Stemt overeen met één teken |

# Benoemingsregels

In deze paragraaf worden de regels besproken voor

- De benoeming van objecten
- De benoeming van methodes, procedures, variabelen en arrays

## Objecten benoemen

In de voorbeelden uit de vorige paragraaf werden objecten gebruikt die expliciet werden benoemd bij het ontwerpen van het formulier. Er werd bijvoorbeeld een kader ontworpen dat *kaderEen* werd genoemd. U hebt waarschijnlijk echter opgemerkt dat Paradox elk object een standaardnaam geeft als u het object maakt (standaardnamen beginnen altijd met een #-teken, bijvoorbeeld *#Kader5*). Als u een object maakt en de standaardnaam niet verandert, hoeft u de naam van dat object niet op te geven in het pad van ingesloten objecten.

***Opmerking*** Het object is onderdeel van de hiërarchie van ingesloten objecten voor eigen code en variabelen, maar de naam van het object is optioneel.

Zie het *Handboek* voor benoemingsregels voor Paradox-objecten.

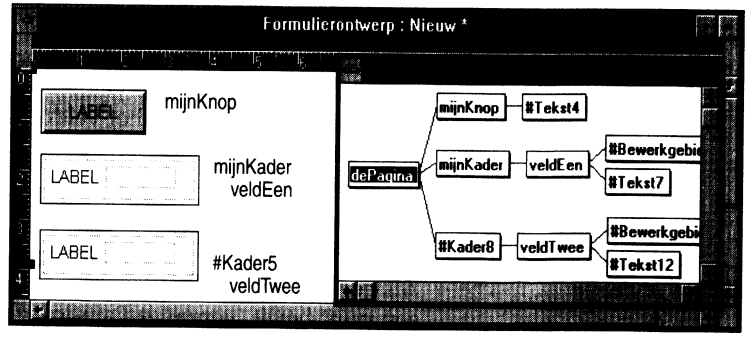
### *Standaardnamen*

In Afbeelding 11-3 ziet u een pagina met één knop (met de naam *mijnKnop*) en twee kaders. Eén kader heet *mijnKader* en het andere kader heeft de standaardnaam *#Kader5*. In elk kader is een veldobject geplaatst (*veldEen* en *veldTwee* genaamd).

Afbeelding 11-3 Standaardnamen en de hiërarchie van ingesloten objecten

Methodes die zijn gekoppeld aan *mijnKnop*, moeten het pad via *mijnKader* opgeven om *veldEen* te adresseren

Methodes die zijn gekoppeld aan *mijnKnop*, kunnen *#Kader5* overslaan en *veldTwee* direct adresseren, omdat *#Kader5* een standaardnaam is




De instructies die aan *mijnKnop* zijn gekoppeld, moeten het pad via *mijnKader* opgeven om *veldEen* te bereiken (als *veldEen* geen unieke naam is), bijvoorbeeld als volgt:

```
mijnKader.veldEen.value = 123
```

De instructies kunnen *#Kader5* ook overslaan en *veldTwee* direct adresseren, zoals:

```
veldTwee.value = 755
```

Omdat *#Kader5* de standaardnaam van het object is, is deze niet nodig in het pad van ingesloten objecten. Deze eigenschap is handig als u objecten puur voor de sier op een formulier wilt plaatsen zonder deze te benoemen en te adresseren om de objecten te bereiken die het eigenlijke werk doen.

***Opmerking*** U kunt een object met behulp van de standaardnaam adresseren, bijvoorbeeld:

```
#kader5.color = Red
```

## Met tabellen verbonden objecten

Als u een object plaatst dat een standaardnaam uit een tabel krijgt, maakt deze naam deel uit van het pad van ingesloten objecten. Alleen standaardnamen die beginnen met een #-teken, kunnen uit het pad worden weggelaten.

Stel dat u een tabelframe op een formulier plaatst en het zo definieert dat het gegevens uit KLANT.DB weergeeft. In Paradox-termen is het tabelframe dan "verbonden" met de tabel *Klant*. Het tabelframe krijgt standaard de naam *KLANT* en de veldobjecten in het tabelframe krijgen de namen van de velden in de tabel *Klant* (zonder #-tekens).

***Belangrijk*** Objectnamen kunnen niet met een cijfer beginnen, zelfs niet als het object met een tabel is verbonden. Een tabel kan bijvoorbeeld een veld hebben met de naam *3Dimensie*, maar een object dat met dat veld is verbonden, kan dat niet. Paradox voorkomt dat u een object

een ongeoorloofde naam geeft in een ontwerpvenster. Als een ObjectPAL-instructie een ongeoorloofde naam toewijst aan een object, vervangt Paradox de ongeldige tekens met onderstrepingstekens.

Als een instructie in een ander object (bijvoorbeeld in een knop) de waarde van een van deze velden probeert in te stellen, moet in het pad het tabelframe vóór het veldobject worden opgegeven. De volgende instructie stelt bijvoorbeeld de waarde van het veld 'Achternaam' in op "Meisner".

```
KLANT.Achternaam.value = "Meisner"
```

### Punten in namen

Als een object de naam krijgt van een veld in een tabel en die naam een punt bevat, vervangt Paradox de punt door een onderstrepingsteken. Stel dat de tabel *Onderdln* een veld bevat met de naam "Onderdeelnr." Op een formulier zou dat veldobject de naam *Onderdeelnr_* krijgen. U stelt de waarde van het object als volgt in:

```
ONDERDLN.Onderdeelnr_.value = "AB123-55"
```

**Opmerking** Paradox verandert de namen in de onderliggende tabel niet.

## Methodes, procedures, variabelen en arrays

Hieronder volgen de regels voor de benoeming van eigen methodes en de ObjectPAL-variabelen, -arrays en -procedures die u in methodes gebruikt:

- Namen mogen maximaal 32 tekens lang zijn.
- Het eerste teken moet een letter zijn, A–Z of a–z.
- De volgende tekens mogen letters, getallen of een van de volgende drie tekens zijn: $ ! _ . Uitgebreide tekens (ANSI 161–255) zijn toegestaan.
- Spaties of tabs zijn niet toegestaan.
- Er wordt geen onderscheid gemaakt tussen hoofdletters en kleine letters.
- Een naam mag niet overeenkomen met een sleutelwoord, een elementair taalelement, een ingebouwde objectvariabele of de naam van objecttypes, acties, methodes of kenmerken.

In Tabel 11-8 worden voorbeelden gegeven van geldige en ongeldige namen van variabelen.

Tabel 11-8 Enkele geldige en ongeldige namen van variabelen

| Naam | Geldig? | Uitleg |
|---|---|---|
| allEs | Ja | Geen onderscheid tussen hoofdletters en kleine letters |
| n.aam | Nee | Bevat "." |
| !claim | Nee | Begint niet met een letter |
| claim! | Ja | Begint met een letter |
| voilà | Ja | Uitgebreide tekens zijn toegestaan |
| L0123 | Ja | Alles in hoofdletters mag en de naam begint met een letter |
| 5A6 | Nee | Begint niet met een letter |
| abc xyz | Nee | Bevat een spatie |
| abc_xyz | Ja | Bevat een onderstrepingsteken in plaats van een spatie |
| record | Nee | Is een sleutelwoord |
| mijnRecord | Ja | Is geen sleutelwoord |

Deze regels hebben alleen betrekking op de namen van ObjectPAL-variabelen, arrays, eigen methodes en eigen procedures.

In theorie mag u geen namen van objecttypes, namen van elementaire taalonderdelen, sleutelwoorden en namen van kenmerken of methodes gebruiken om objecten of variabelen te benoemen. In de praktijk is dit niet altijd mogelijk. Een applicatie voor een internationale verffabriek kan bijvoorbeeld een tabel bevatten waarin zich een veld bevindt met de naam 'Color'. Gebruik, om verwarring met het kenmerk 'Color' te voorkomen, één aanhalingsteken in plaats van een punt vóór de naam van het kenmerk. (In de Nederlandse Paradox voor Windows heet dit kenmerk 'Kleur', maar in de ObjectPAL-code worden voor kenmerken de Engelse namen gebruikt.) In het volgende voorbeeld wordt het kenmerk 'Color' van het veld 'Prijs' ingesteld:

```
prijs.Color = Red ; gebruikt een punt
```

In het volgende voorbeeld wordt het kenmerk 'Color' van het veldobject *Color* ingesteld. (De veldnaam staat vóór het aanhalingsteken en de naam van het kenmerk komt erachter.)

```
Color'Color = Red ; gebruikt een enkel aanhalingsteken
```

U kunt ook twee enkele aanhalingstekens gebruiken (*niet* één dubbel aanhalingsteken) in plaats van het sleutelwoord *self* vóór de kenmerknaam. De volgende twee instructies zijn bijvoorbeeld gelijk:

```
''Color = Red

self.color = Red
```

Hoewel u een object een naam kunt geven die begint met "#", is dit niet aan te bevelen. De standaardnamen die Paradox aan ontwerpobjecten geeft, beginnen altijd met "#" (bijvoorbeeld *#knop1320*). Het is verstandig de objecten die u hebt benoemd, te onderscheiden van de objecten die u niet hebt benoemd.

# ObjectPAL-termen

De termen *methode, procedure* en *elementair taalonderdeel* hebben speciale betekenissen in ObjectPAL, zoals in de volgende paragrafen wordt uitgelegd. In Tabel 11-11, aan het einde van het hoofdstuk, wordt een overzicht gegeven van de verschillen.

## Methodes

Een methode is code die het gedrag van een object definieert. Methodes definiëren hoe een object op acties reageert. Deze acties kunnen worden gegenereerd door gebruikers, door Paradox en zelfs door andere methodes. De ObjectPAL-methodes vallen in drie categorieën uiteen:

- Ingebouwde methodes die in elk Paradox-object zijn opgenomen
- Methodes in de runtime bibliotheek (RTL: runtime library) van ObjectPAL
- Eigen methodes die u zelf maakt

De kenmerken van de verschillende categorieën methodes worden weergegeven in Tabel 11-9.

Tabel 11-9 Eigenschappen van methodes

| Ingebouwd | RTL | Eigen |
|---|---|---|
| Ingebouwde objecten op een formulier | Gebruikt voor het schrijven van eigen code | Een door de gebruiker gedefinieerde routine |
| Geeft standaardgedrag van objecten aan | Werkt op objecten van een specifiek type | Gekoppeld aan een object |
| Wordt automatisch uitgevoerd in reactie op acties | Wordt binnen een ObjectPAL-instructie uitgevoerd | Wordt uitgevoerd indien aangeroepen door een ObjectPAL-instructie |
| Kan worden gewijzigd door koppeling van eigen code | Kan worden gebruikt in code die is gekoppeld aan een object | Globaal: kan met behulp van puntnotatie door andere objecten worden aangeroepen |
| | Puntnotatie geeft aan op welk object wordt gewerkt | |

ObjectPAL-methodes maken de taal symmetrisch en consistent: binnen een type komen methodes vaak in paren voor. Als een type

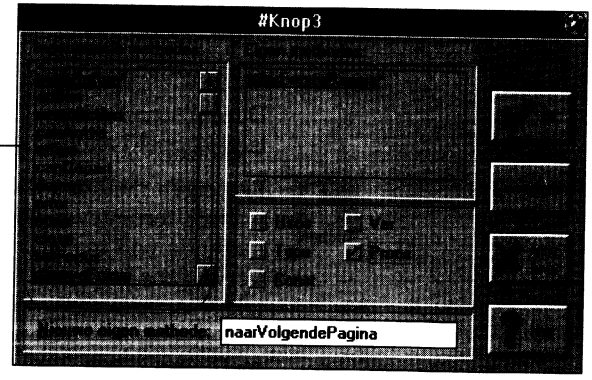
bijvoorbeeld een **open**-methode heeft, kunt u ervan uitgaan dat het ook een **close**-methode heeft. Als u informatie uit een object kunt lezen, kunt u er ook naar schrijven; als u een waarde kunt ophalen, kunt u deze ook instellen. ObjectPAL-methodes voor verschillende types komen met elkaar overeen, omdat methodes met dezelfde namen dezelfde dingen doen De **open**-methode maakt een object bijvoorbeeld beschikbaar voor manipulatie, ongeacht of het object een tabel is of een tekstbestand. De **close**-methode zet het object weer weg. De onderliggende code kan verschillen, maar de resultaten zijn conceptueel gezien hetzelfde.

### Ingebouwde methodes

Elk Paradox-object heeft ingebouwde methodes (bijvoorbeeld **open**, **close** en **mouseUp**) voor elke actie waarop het kan reageren. Deze ingebouwde methodes geven het standaardgedrag aan van een object, in reactie op een bepaalde actie. U maakt Paradox-applicaties door ObjectPAL-code te koppelen aan de ingebouwde methodes van objecten die in formulieren zijn geplaatst. U koppelt uw eigen code aan ingebouwde methodes met behulp van de ObjectPAL-Editor. Als u een ingebouwde methode van een object wilt bewerken, inspecteert u het object en kiest u 'Methodes' in het kenmerkmenu. Selecteer één of meer methodes in het methodevenster (zie Afbeelding 9-1) en kies vervolgens 'OK' om één of meer vensters van de ObjectPAL-Editor te openen. U kunt de tekst voor een methode direct in de ObjectPAL-Editor typen of het Klembord gebruiken om methodes en gedeelten van methodes te kopiëren, te knippen en te plakken vanuit andere objecten of vanuit een bestand.

**Opmerking** Alle ingebouwde methodes worden beschreven in Appendix B.

Afbeelding 11-4 Ingebouwde methodes selecteren voor bewerking




Als u ingebouwde methodes wilt bewerken, kiest u één of meer opties in deze lijst. In deze afbeelding is de methode **pushButton** gekozen.

### Methodes in de runtime bibliotheek

De runtime bibliotheek (RTL) van ObjectPAL is een verzameling voorgedefinieerde routines. De runtime bibliotheek bevat methodes waarmee u een groot aantal uiteenlopende taken kunt uitvoeren: van

het lezen en bewerken van gegevens in tabellen tot het maken en weergeven van menu's. Elke methode uit de bibliotheek is geassocieerd met een objecttype: methodes voor het werken met formulieren bevinden zich in het Form-type, methodes voor het werken met tekstbestanden bevinden zich in het TextStream-type enzovoort. U vindt een beschrijving van deze methodes in de online ObjectPAL Help.

Bij methodes in de runtime bibliotheek dient u de puntnotatie te gebruiken om een object op te geven waarop u wilt werken. Bijvoorbeeld:

```
var
   bestel, verkoopF Form
   verkoopTV TableView
endVar
bestel.open("bestel")
verkoopF.open("verkoop")
verkoopF.setTitle("Verkoopinfo")  ; stel de naam van het verkoopformulier in
bestel.maximize()                 ; vergroot bestel, doe niets met verkoop
verkoopTV.open("verkoop.db")      ; open een tabelvenster voor het
verkoopformulier
```

In dit voorbeeld worden de objecten (*verkoopF*, *verkoopTV* en *bestel*) door puntnotatie gescheiden van de methodes (**open, setTitle** en **maximize**).

## Eigen methodes

Eigen methodes zijn hulproutines die u zelf maakt. Deze methodes zijn handig om veel gebruikte routines beschikbaar te maken voor verschillende objecten. Eigen methodes zijn globaal; dat wil zeggen, eigen methodes die zijn gekoppeld aan een object, kunnen ook door een ander object worden aangeroepen. (In tegenstelling tot eigen procedures, die lokaal zijn.) Stel dat een formulier twee kaders bevat: *kader1* en *kader2*. Als *kader1* een eigen methode bevat met de naam **doeIets**, kan code die is gekoppeld aan *kader2* de methode met de volgende instructie aanroepen:

```
kader1.doeIets()
```

Deze instructie zegt: "Voer de methode uit met de naam **doeIets** die is gekoppeld aan het object *kader1*".

Een eigen methode kan één of meer argumenten bevatten en kan een waarde teruggeven (maar dit is niet noodzakelijk). Als u een eigen methode wilt maken, inspecteert u een object en kiest u 'Methodes'. Vervolgens kiest u het tekstvak 'Nieuwe eigen methode'. Typ in het tekstvak een naam voor de eigen methode. Kies vervolgens 'OK' om een venster van de ObjectPAL-Editor te openen. U kunt tekst in eigen methodes typen of plakken, zoals dat ook bij ingebouwde methodes kan.

Als u een eigen methode hebt opgeslagen, wordt de naam in het methodevenster weergegeven. Als u wijzigingen wilt aanbrengen,

kiest u de naam en opent u een venster van de ObjectPAL-Editor, zoals u dat ook bij een ingebouwde methode doet.

**Opmerking**
*Eigen methodes die aan het formulier zijn gekoppeld, zijn voor alle objecten op het formulier beschikbaar.*

Als u methodes aan een formulier wilt koppelen, kiest u 'Kenmerken | Formulier | Methodes'. U kunt ook het Objectenschema gebruiken. Zie de uitleg van het Objectenschema in Hoofdstuk 9 voor meer informatie. Een andere snelle manier: druk op *Ctrl-Spatiebalk* als er geen objecten zijn geselecteerd.

Als u eigen code in twee of meer formulieren gezamenlijk wilt gebruiken, plaatst u de code in een bibliotheek. Zie Hoofdstuk 17 voor meer informatie.

## Procedures

Er zijn twee soorten procedures in ObjectPAL:

- Procedures in de runtime bibliotheek (RTL) van ObjectPAL
- Eigen procedures die u zelf maakt

In Tabel 11-10 wordt een overzicht gegeven van de kenmerken van deze procedures.

Tabel 11-10 Eigenschappen van ObjectPAL-procedures

| RTL | Eigen |
|---|---|
| Gebruikt voor het schrijven van eigen code | Een door de gebruiker gedefinieerde routine |
| Werkt op objecten van een bepaald type | Gedefinieerd in het Proc-venster van het object |
| Kan worden gebruikt in code die is gekoppeld aan een object | |
| Geeft niet aan op welk object wordt gewerkt; het object is impliciet | Lokaal: kan niet met puntnotatie door andere objecten worden aangeroepen |
| Wordt binnen een ObjectPAL-instructie uitgevoerd | Wordt uitgevoerd indien aangeroepen door een ObjectPAL-instructie |

### RTL-procedures

De procedures in de runtime bibliotheek (RTL) van ObjectPAL werken hetzelfde als methodes, met één uitzondering: een object wordt nooit expliciet opgegeven door een procedure. Code die is gekoppeld aan een object, kan elke RTL-procedure aanroepen. De procedure weet dan wat er moet gebeuren. RTL-procedures zijn net als RTL-methodes geassocieerd met andere objecttypes, en worden uitgevoerd in reactie op een actie. U kunt RTL-procedures beschouwen als RTL-methodes waarbij het object impliciet is.

De instructie `close()` roept bijvoorbeeld de procedure **close** voor het type Form aan, die het huidige formulier sluit. Het huidige formulier

hoeft niet met puntnotatie te worden opgegeven, omdat het door de procedure wordt geïmpliceerd.

Andere procedures zetten vlaggen die betrekking hebben op het systeem. De **blankAsZero**-procedure van het Session-type geeft bijvoorbeeld aan hoe lege waarden worden afgehandeld.

Het System-type bevat procedures om dialoogvensters weer te geven. U gebruikt deze procedures zonder een object op te geven. Bijvoorbeeld:

```
msgStop("Let op!", "Dit bestand bestaat al.")
```

Het System-type bevat ook de procedures **beep** en **sleep** en verschillende procedures om de positie en de vorm van de aanwijzer in te stellen. De **message**-procedure is ook gedefinieerd voor het System-type, net als verschillende procedures die informatie over ObjectPAL naar Paradox-tabellen schrijven. In het volgende voorbeeld worden de procedures **beep**, **sleep**, **message** en **enumSource** gebruikt.

```
method pushButton(var eventInfo Event)
   beep()                              ; laat de pieptoon van het systeem horen
   sleep(2000)                         ; wacht twee seconden
   beep()
   message("Hoorde u twee pieptonen?") ; geeft een bericht op de statusregel
                                       ; weer
   sleep(2000)
   enumSource("mijnBron.db")           ; maakt een tabel van alle code op dit
                                       ; formulier
endMethod
```

## Eigen procedures

Eigen procedures in ObjectPAL lijken op de procedures van veel andere programmeertalen. Een eigen procedure is een routine die u zelf schrijft en als een subroutine gebruikt. Eigen procedures zijn lokaal; de beschikbaarheid van deze procedures voor andere objecten wordt bepaald door insluiting (dit wordt in de volgende paragraaf beschreven).

***Belangrijk*** Paradox kan een eigen procedure sneller aanroepen dan een eigen methode. De code wordt even snel uitgevoerd, maar vanwege de manier waarop een eigen procedure wordt opgeslagen, kan Paradox deze sneller "vinden" dan een eigen methode.

U kunt procedures op twee plaatsen declareren:

- Binnen een methode
- In het Proc-venster van een object

De syntaxis voor de definitie van procedures vindt u in de beschrijving van **Proc** in de online ObjectPAL Help.

*Procedures die in methodes zijn gedeclareerd*

Een procedure die in een methode is gedeclareerd, is lokaal: deze procedure kan alleen worden aangeroepen door de methode waarin

de procedure is gedeclareerd. Hieronder volgt een voorbeeld van een eigen procedure:

```
proc plus (x SmallInt) SmallInt
   return x+1 ; verhoogt een getal
endProc
```

In het volgende voorbeeld ziet u hoe deze procedure (en nog een andere) wordt gedeclareerd en aangeroepen vanuit een methode. (In dit voorbeeld gaat het om de **pushButton**-methode, maar het kan ook een andere methode zijn). De procedures worden eerst gedefinieerd, vóór het hoofddeel van de methode.

```
   proc plus (x SmallInt) SmallInt
      return x+1
   endProc
   proc toonMij (x SmallInt)
      msgInfo("mijnGetal = ", x)
   endProc
method pushButton (var eventInfo Event)
   var
      mijnGetal SmallInt
   endVar
   mijnGetal = 3
   toonMij (mijnGetal)
   mijnGetal = plus(mijnGetal)
   toonMij (mijnGetal)
endMethod
```

*Procedures die in het Proc-venster van een object zijn gedeclareerd*

Een procedure die in het Proc-venster van een object wordt gedeclareerd, heeft dezelfde syntaxis als een procedure die in een methode wordt gedeclareerd. Het gebruik van die procedure is echter anders. Een procedure die in het Proc-venster van een object wordt gedeclareerd, is zichtbaar voor alle methodes die aan dat object zijn gekoppeld, en voor alle methodes in objecten die zich in *dat* object bevinden. In tegenstelling tot een eigen methode, kunt u een procedure echter niet aanroepen van buiten de hiërarchie van ingesloten objecten van de procedure.

Als u een eigen procedure beschikbaar wilt maken voor alle objecten op een formulier, declareert u de procedure in het Proc-venster van het formulier.

Als u code aan een formulier wilt koppelen, kiest u 'Kenmerken | Formulier | Methodes'. U kunt ook het Objectenschema gebruiken. Zie de uitleg van het Objectenschema in Hoofdstuk 9 voor meer informatie.

## Elementaire taalonderdelen

Elementaire taalonderdelen zijn controlestructuren, zoals **if...then... else** en **switch...endSwitch**, opdrachten, zoals **quitLoop** en sleutelwoorden voor het maken van structuren, zoals **var...endVar**. Hierbij wordt geen puntnotatie gebruikt. De syntaxis voor de structuur **for...endFor** luidt bijvoorbeeld:

**for** *VarNaam* [**from** *startWaarde*] [**to** *eindWaarde*] [**step** *stapWaarde*]
*Instructies*
**endFor**

In de online ObjectPAL Help worden de elementaire taalonderdelen besproken.

## Overzicht van verschillen

In Tabel 11-11 wordt een overzicht gegeven van de verschillen tussen RTL-methodes en eigen methodes, RTL-procedures en eigen procedures en elementaire taalonderdelen. Daarnaast worden in de tabel prototypes getoond (elementen tussen vierkante haken zijn optioneel) en wordt aangegeven of een constructie globaal is. In de prototypes vertegenwoordigt *obj* een objectnaam of een pad van ingesloten objecten en *argLijst* een lijst van één of meer argumenten en types van teruggegeven waarden.

Tabel 11-11 Overzicht van verschillen

| Constructie | Prototype | Globaal? |
|---|---|---|
| RTL-methode | obj.methode([*argLijst*]) [*terugType*] | Ja |
| Eigen methode | [obj].methode([*argLijst*]) [*terugType*] | Ja |
| RTL-procedure | procNaam([*argLijst*]) [*terugType*] | Ja |
| Eigen procedure | procNaam([*argLijst*]) [*terugType*] | Nee |
| Elementair taalonderdeel | sleutelwoord (return), of structuur (for...endFor) | NVT |

# Insluitende objecten

Het gedrag van een ontwerpobject wordt bepaald door de methodes, de kenmerken en de visuele context van het object. De methodes zijn de ingebouwde methodes en de eigen code die u schrijft. De kenmerken zijn eigenschappen die interactief worden ingesteld tijdens het ontwerpen of door ObjectPAL-instructies worden ingesteld tijdens de uitvoering. De insluitrelaties—de visuele, ruimtelijke relaties van een object met andere objecten—verschaffen de context.

Alle objecten op een formulier zijn onderverdeeld in een hiërarchie van insluitende objecten. Als u bijvoorbeeld een tabelframe op een pagina van een formulier plaatst, sluit het formulier de pagina in en sluit de pagina de tabel in. De positie in deze hiërarchie is belangrijk, omdat hierdoor wordt bepaald welke methodes, procedures, kenmerken en variabelen een object van andere objecten kan krijgen.

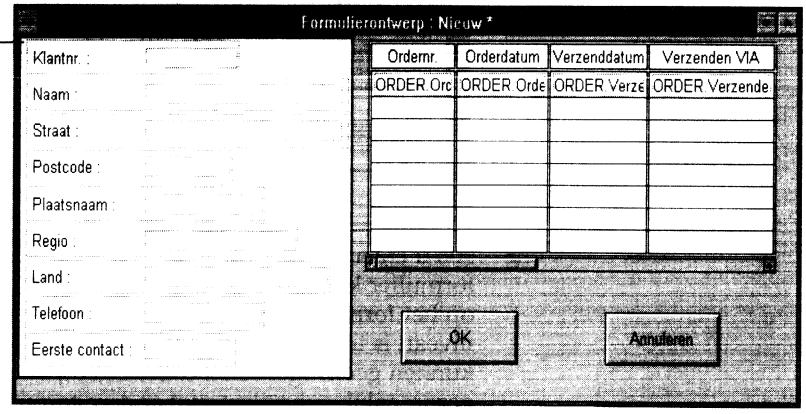
De beschikbaarheid van dergelijke hulpbronnen hangt samen met wat u op het scherm ziet.

***Belangrijk*** Een object is ingesloten als het zich *geheel binnen* de grenzen van het insluitend object bevindt. (Objecten die verschuiven, bijvoorbeeld veldobjecten in een tabelframe, zijn ook ingesloten.) Als u het kenmerk 'Objecten insluiten' van een object deselecteert, wordt dat object door ObjectPAL echter niet als een insluitend object behandeld.

Een object heeft directe toegang tot de eigen methodes, procedures, kenmerken en variabelen en tot de methodes, procedures, kenmerken en variabelen die zijn gedeclareerd in de objecten die het object insluiten. Stel dat een formulier bestaat uit een groep veldobjecten, zoals in Afbeelding 11-5 en dat u wilt dat er in de veldobjecten een prompt verschijnt als de gebruiker de invoegpositie erin plaatst. In plaats van code te koppelen aan elk veldobject, kunt u een kader tekenen rondom deze veldobjecten en de code koppelen aan het kader. De veldobjecten hebben directe toegang tot eigen methodes en procedures die zijn gekoppeld aan het kader, omdat de veldobjecten zich in het kader bevinden.

Afbeelding 11-5 Een insluitend object gebruiken om gezamenlijk gebruikte code op te slaan

Omdat het kader de veldobjecten insluit; is de eigen code die aan het kader is gekoppeld, beschikbaar voor alle veldobjecten




Stel dat een pagina op een formulier twee knoppen insluit, zoals in Afbeelding 11-6. Alle variabelen die in het Var-venster van de pagina zijn gedeclareerd, zijn voor beide knoppen beschikbaar. U kunt dus bijvoorbeeld de volgende code koppelen om een Number-variabele te declareren en aan deze variabele een waarde toe te wijzen. Vervolgens gebruikt u de knoppen om de waarde van de variabele te verhogen of te verlagen:

*pagina::Var-venster*

```
; pagina::Var-venster
Var
```

```
   hetGetal Number
endVar
```

*pagina::open-methode*

```
; pagina::open
method open(var eventInfo Event)
   hetGetal = 0
endmethod
```

*knopEen::pushButton-methode*

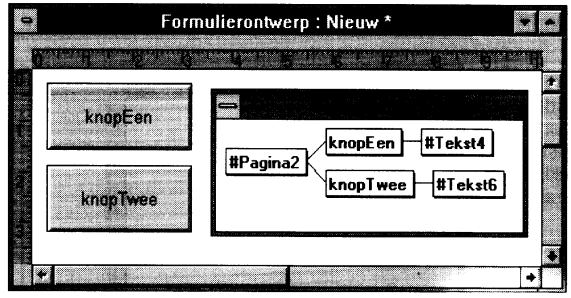
```
; knopEen::pushButton
method pushButton(var eventInfo Event)
   hetGetal = hetGetal + 1
   message(hetGetal)
endmethod
```

*knopTwee::pushButton-methode*

```
; knopTwee::pushButton
method pushButton(var eventInfo Event)
   hetGetal = hetGetal - 1
   message(hetGetal)
endmethod
```

Omdat de variabele *hetGetal* in het Var-venster van de pagina is gedeclareerd, kan elk object dat zich op de pagina bevindt, toegang krijgen tot *hetGetal*.

Afbeelding 11-6 Ingesloten objecten en variabelen




De onderliggende pagina sluit twee knoppen in, waardoor een variabele die in het Var-venster van de pagina is gedeclareerd, beschikbaar is voor beide knoppen

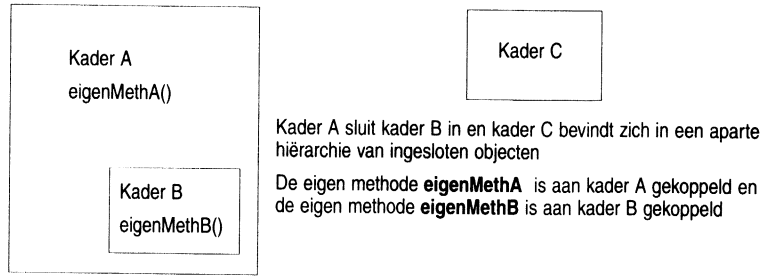
***Tip*** Als u eigen methodes, eigen procedures en variabelen aan het formulier koppelt, maakt u die beschikbaar voor alle objecten die zich op het formulier bevinden, omdat het formulier het hoogste object-niveau is in de hiërarchie van ingesloten objecten. Formulieren kunnen geen eigen methodes, eigen procedures of variabelen gezamenlijk gebruiken met andere formulieren, omdat er geen insluitend object op een hoger niveau is waaraan u die formulieren kunt koppelen. (U kunt echter eigen code in een bibliotheek opslaan en de bibliotheek beschikbaar maken voor meerdere formulieren. Zie de paragraaf "Library" van Hoofdstuk 17 voor meer informatie.)

## Insluiting en eigen methodes

Insluitrelaties hebben invloed op de beschikbaarheid van eigen methodes. Een object heeft directe toegang (zonder puntnotatie) tot de eigen methodes van het object en tot de methodes in objecten die zich in het object bevinden. Aangezien eigen methodes globaal zijn, kunt u bovendien puntnotatie gebruiken om eigen methodes aan te

roepen die zijn gekoppeld aan objecten in andere hiërarchieën van ingesloten objecten. In Afbeelding 11-7 ziet u bijvoorbeeld drie kaders, *A*, *B* en *C*, waarbij kader *A* kader *B* insluit.

Afbeelding 11-7 Insluiting en eigen methodes




Vanwege de hiërarchie van ingesloten objecten in Afbeelding 11-7, zijn de volgende beweringen waar:

- Kader *B* kan de eigen methode **eigenMethB** direct aanroepen, omdat **eigenMethB** is gekoppeld aan kader *B*:

  ```
  eigenMethB()
  ```

- Kader *B* kan de eigen methode **eigenMethA** ook direct aanroepen, omdat **eigenMethA** is gekoppeld aan kader *A*, dat kader *B* insluit:

  ```
  eigenMethA()
  ```

- Kader *C* kan puntnotatie gebruiken om **eigenMethB**, die is gekoppeld aan kader *B*, aan te roepen:

  ```
  b.eigenMethB()
  ```

- Kader *C* kan puntnotatie gebruiken om **eigenMethA**, die is gekoppeld aan kader *A*, aan te roepen:

  ```
  a.eigenMethA()
  ```

- Kader *A* moet puntnotatie gebruiken om **eigenMethB** aan te roepen:

  ```
  b.eigenMethB()
  ```

- Kader *C* kan de volgende aanroep doen vanwege de hiërarchie van ingesloten objecten:

  ```
  b.eigenMethA()
  ```

In het laatste geval zoekt Paradox in de code die is gekoppeld aan kader *B* naar **eigenMethA**. Als deze methode niet wordt gevonden, wordt verder gezocht in het insluitend object van *B*. Als Paradox constateert dat **eigenMethA** is gekoppeld aan kader *A*, wordt de code

uitgevoerd. Anders wordt steeds hoger in de hiërarchie gezocht, te beginnen met het insluitend object van kader *A*. Als een eigen methode wordt aangeroepen, bepaalt de puntnotatie niet noodzakelijkerwijs met welk object wordt gewerkt. De puntnotatie bepaalt echter wel in welk object naar de code wordt gezocht.

***Opmerking*** Als twee of meer objecten op een formulier dezelfde naam hebben, gebruikt u puntnotatie om op te geven met welk object u wilt werken.

### Subject

De objectvariabele *Subject* geeft aan op welk object een eigen methode moet werken. Als een eigen methode wordt aangeroepen, bepaalt de puntnotatie welk object het subject is van de methode. Stel, uitgaande van het vorige voorbeeld, dat de code voor de eigen methode **eigenMethA**, die is gekoppeld aan kader *A*, als volgt luidt:

*Gekoppeld aan kader A*

```
method eigenMethA()
   Subject.color = Red
endmethod
```

Als kader *C* de instructie `b.eigenMethA()` maakt, wordt kader *B* rood, omdat kader *B* het subject is.

Als kader *C* de instructie `a.eigenMethA()` maakt, wordt kader *A* rood, omdat kader *A* het subject is.

Als kader *B* de instructie `eigenMethA()` maakt, wordt kader *B* rood, omdat Paradox kader *B* herkent als het subject van de methode.

***Belangrijk*** *Subject* is niet hetzelfde als *Self*, een andere objectvariabele. Het subject wordt bepaald door de puntnotatie, terwijl *Self* altijd het object is waaraan de code is gekoppeld. Stel dat de code voor de eigen methode **eigenMethB** wordt veranderd in het volgende:

*Gekoppeld aan kader B*

```
method eigenMethB()
   self.color = Red
endmethod
```

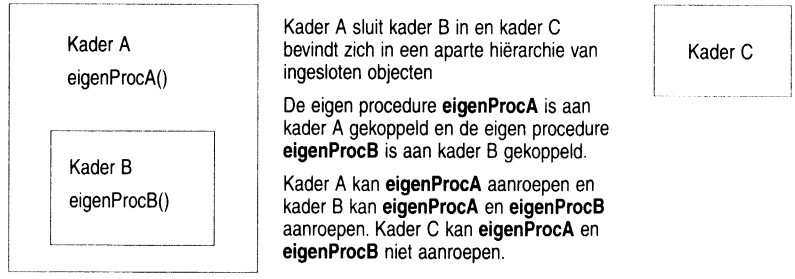
In dit geval verandert kader *B* altijd van kleur als een object **eigenMethB** aanroept, ongeacht de puntnotatie, omdat **eigenMethB** is gekoppeld aan kader *B* en *Self* altijd het object is waaraan de code is gekoppeld. Raadpleeg Hoofdstuk 13 voor meer informatie over het gebruik van *Self*.

## Insluiting en eigen procedures

Insluitrelaties hebben invloed op de beschikbaarheid van eigen procedures. Een object heeft toegang tot de eigen procedures en tot de procedures die zijn gedefinieerd in objecten die het object insluiten. Eigen procedures zijn lokaal. Dit houdt in dat u geen eigen procedures kunt aanroepen die zijn gekoppeld aan objecten die zich in andere hiërarchieën van ingesloten objecten bevinden. In

Afbeelding 11-8 ziet u bijvoorbeeld drie kaders, *A*, *B* en *C*, waarbij kader *A* kader *B* insluit.

Afbeelding 11-8 Insluiting en eigen procedures




Vanwege de hiërarchie van ingesloten objecten in Afbeelding 11-8, zijn de volgende beweringen waar:

- Kader *B* kan de eigen procedure **eigenProcB** aanroepen, omdat **eigenProcB** is gekoppeld aan kader *B*:

  ```
  eigenProcB()
  ```

- Kader *B* kan de eigen procedure **eigenProcA** ook aanroepen, omdat **eigenProcA** is gekoppeld aan kader *A*, dat kader *B* insluit:

  ```
  eigenProcA()
  ```

- Kader *C* kan **eigenProcB**, die is gekoppeld aan kader *B*, niet aanroepen.
- Kader *C* kan **eigenProcA**, die is gekoppeld aan kader *A*, niet aanroepen.
- Kader *A* kan **eigenProcB**, die is gekoppeld aan kader *B*, niet aanroepen.

# Variabelen

Een *variabele* is een opslagplaats waarin u tijdelijk één stukje informatie kunt opslaan. De waarde van een variabele kan van elk ObjectPAL-type (ook wel *gegevenstype* genoemd) zijn.

De eenvoudigste manier om een variabele een waarde te geven, is met behulp van de toewijzingsoperator (=). De volgende instructie wijst bijvoorbeeld de reeks "abc" toe aan de variabele *x*:

```
x = "abc"
```

Als aan *x* al eerder een waarde is toegewezen, gaat deze waarde verloren.

U moet een waarde aan een variabele toewijzen voordat u de variabele in een uitdrukking gebruikt. De volgende instructie wordt niet uitgevoerd als er geen waarde aan *x* is toegewezen:

```
mijnBericht = "De waarde van x is " + strVal(x)
```

U kunt **isAssigned** gebruiken om te testen of er een waarde aan een variabele is toegewezen. U doet dit als volgt:

```
if mijnVar.isAssigned() = True then
   mijnVar = mijnVar + 1
else
  mijnVar = 1
endIf
```

U hoeft niet expliciet een gegevenstype op te geven voor variabelen: ObjectPAL gaat ervan uit dat u een van de gegevenstypes uit Tabel 11-1 wilt gebruiken. De volgende instructies maken bijvoorbeeld van *x* eerst een String-variabele, vervolgens een SmallInt en dan een Number.

```
x = "abc"       ; x is een String
x = 5           ; x is een SmallInt
x = 3.238       ; x is een Number
```

Als u complexere variabelen wilt gebruiken (bijvoorbeeld Array of TCursor), dient u deze te declareren. De volgende instructie wordt niet gecompileerd, tenzij de variabele *orderTC* ergens in het bereik van de instructie wordt gedeclareerd:

```
orderTC.open("order.db")
```

Het heeft bovendien voordelen om expliciet te declareren welk variabeletype u gebruikt. Deze voordelen worden besproken in de volgende paragraaf.

## Variabelen declareren

*Een variabele declareren* betekent dat u het variabeletype opgeeft voordat u de variabele gebruikt. Het is verstandig variabelen te declareren. Hieronder volgen enkele voordelen:

- De compiler vangt spelfouten en inconsistent gebruik op.
- De compiler kan uw code optimaliseren, zodat deze sneller wordt uitgevoerd.
- Code "documenteert zichzelf" en wordt leesbaarder.

U kunt variabelen declareren met behulp van de volgende structuur:

```
var
   varNaam1              varType1
   varNaam2a, varNaam2b  varType2
   ; enzovoort
endVar
```

**var** en **endVar** zijn sleutelwoorden. *varNaam1*, *varNaam2a* en *varNaam2b* vertegenwoordigen namen die u voor uw variabelen kiest (variabelen dienen te worden benoemd volgens de regels die eerder in dit hoofdstuk zijn beschreven). *varType1* en *varType2* geven de gegevenstypes aan. Hieronder volgt een voorbeeld:

```
var
   i, j, bedrag   Number
   Voornaam, Achternaam String
endVar
```

Deze code declareert vijf variabelen: *i*, *j* en *bedrag* zijn variabelen van het type Number. *Voornaam* en *Achternaam* zijn variabelen van het type String.

U kunt de compiler opdragen te waarschuwen als ongedefinieerde variabelen worden gebruikt: kies in een Editor-venster 'Kenmerken | Compiler-waarschuwing tonen'.

## Bereik van een variabele

De term "bereik" betekent "beschikbaarheid". Het *bereik* van een variabele, dat wil zeggen, de objecten die toegang hebben tot een variabele, wordt gedefinieerd door het object waarin de variabele is gedeclareerd en door de hiërarchie van ingesloten objecten. Een object heeft alleen toegang tot de eigen variabelen en tot de variabelen die zijn gedeclareerd in het object waarin het zich bevindt. Bovendien is het bereik van een variabele afhankelijk van de plaats waar de variabele is gedeclareerd. U kunt een variabele op de volgende plaatsen declareren:

- Binnen een methode
- Buiten een methode
- In het Var-venster
- Binnen de hiërarchie van ingesloten objecten (verbinding tijdens compilatie)

### *Gedeclareerd binnen een methode*

Variabelen die binnen een methode zijn gedeclareerd, zijn alleen zichtbaar voor die methode en zijn alleen toegankelijk als die methode wordt uitgevoerd. Deze variabelen worden elke keer dat de methode wordt uitgevoerd, geïnitialiseerd (opnieuw ingesteld). In de volgende methode is *mijnGetal* bijvoorbeeld altijd gelijk aan 1, omdat deze variabele elke keer dat de **pushButton**-methode wordt uitgevoerd, wordt gedeclareerd en geïnitialiseerd:

```
method pushButton (var eventInfo Event)
   var
      mijnGetal SmallInt
   endVar
   if mijnGetal.isAssigned() = FALSE then
      mijnGetal = 1
   else
```

```
      mijnGetal = mijnGetal + 1
   endIf

   message(mijnGetal) ; geeft 1 weer
endMethod
```

### Gedeclareerd buiten een methode

Variabelen die in een methodevenster worden gedeclareerd *vóór* het woord **method**, zijn alleen zichtbaar voor die methode, maar worden *niet* geïnitialiseerd als de methode wordt uitgevoerd.

In het volgende voorbeeld wordt *mijnGetal* buiten de **pushButton**-methode gedeclareerd (maar wel binnen hetzelfde methodevenster), waardoor de waarde elke keer met 1 wordt verhoogd als de methode wordt aangeroepen. (Als u code boven de eerste methoderegel wilt invoegen, plaatst u de invoegpositie links van de **m** in **method**. Vervolgens drukt u eenmaal of meerdere malen op *Enter* om witregels in te voegen en typt u de code.)

```
   var
      mijnGetal SmallInt
   endVar

method pushButton (var eventInfo Event)
   if mijnGetal.isAssigned() = FALSE then
      mijnGetal = 1
   else
      mijnGetal = mijnGetal + 1
   endIf

   message(mijnGetal)
endMethod
```

### Gedeclareerd in het Var-venster

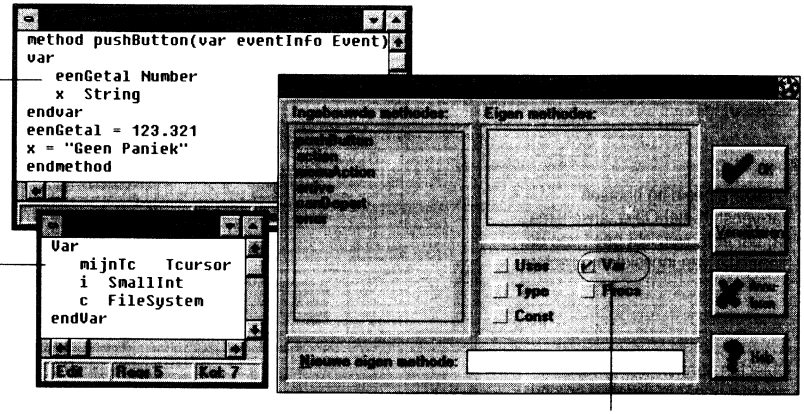
Variabelen die in het Var-venster van een object worden gedeclareerd, zijn zichtbaar voor alle methodes die zijn gekoppeld aan dat object en aan alle objecten die *dat* object insluit. Een variabele die in het Var-venster van een object is gedeclareerd, wordt gekoppeld aan het object en is toegankelijk zolang het object op het formulier aanwezig is en het formulier open is.

Als u variabelen in het Var-venster wilt declareren, inspecteert u een object en kiest u 'Methodes'. Vervolgens kiest u 'Var' en 'OK' om het Var-venster te openen. Het Var-venster is, net als het methodevenster, een venster waarin u tekst bewerkt. In Afbeelding 11-9 ziet u variabelen die in een methodevenster en in een Var-venster zijn gedeclareerd.

Afbeelding 11-9 Variabelen in vensters declareren

Variabelen die in een methode zijn gedeclareerd, zijn alleen voor die methode zichtbaar

Variabelen die in het Var-venster zijn gedeclareerd, zijn zichtbaar voor alle methodes in een object en voor alle objecten die een object insluit.




Klik op 'Var' in het methodevenster om variabelen in het Var-venster van een object te declareren

## *Bereik: een voorbeeld*

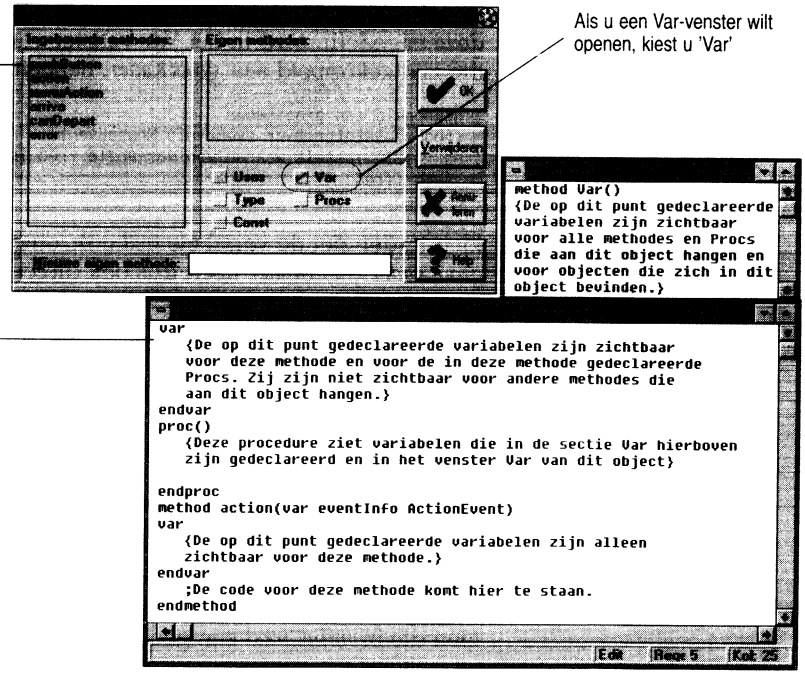
De pseudocode in Afbeelding 11-10 laat zien hoe het bereik van een variabele afhangt van de plaats waar de variabele is gedeclareerd.

Afbeelding 11-10 Bereik van variabelen

Als u een methodevenster wilt openen, zoals het venster dat hier voor **action** wordt getoond, kiest u een methode in deze lijst van ingebouwde methodes (of maakt u een eigen methode).

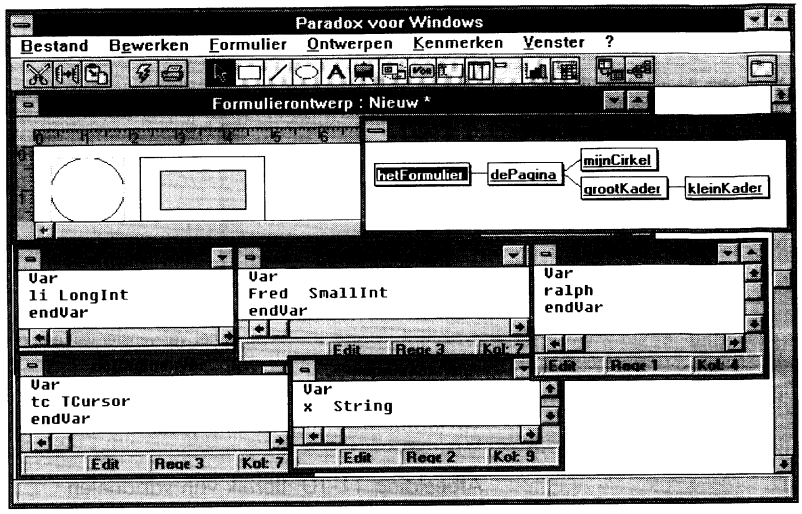
Als u een Var-venster wilt openen, kiest u 'Var'

Dit methodevenster voor **action** laat zien dat het bereik van een variabele afhankelijk is van de plaats van de declaratie



In Afbeelding 11-11 ziet u een formulier met een ellips, *mijnCirkel*, en een rechthoek, *grootKader*, waarin zich nog een rechthoek, *kleinKader*, bevindt.

Afbeelding 11-11 De hiërarchie van ingesloten objecten




In deze afbeelding bootsen Var-vensters het Objectenschema na, dat een overzicht geeft van de hiërarchie van ingesloten objecten

Objecten kunnen wel variabelen zien die hoger in dezelfde sprong zijn gedeclareerd, maar niet variabelen die zijn gedeclareerd in andere sprongen. *kleinKader* kan bijvoorbeeld wel *fred* zien die is gedeclareerd in *grootKader*, en *tc* die is gedelcareerd in *dePagina*, maar niet *x* die is gedeclareerd in *mijnCirkel*.

In Afbeelding 11-11 kan grootKader de variabele *fred* zien, omdat deze is gedefinieerd in het Var-venster van *grootKader*. De methodes die zijn gekoppeld aan *grootKader*, hebben direct toegang tot *fred*:

```
; gekoppeld aan grootKader
method mouseEnter (var eventInfo MouseEvent)
fred = 123  ; stelt waarde van variabele fred in op 123
message (fred)
endMethod
```

*Een object heeft directe toegang tot variabelen die zijn gedeclareerd in objecten die het insluiten*

kleinKader ziet de variabele *fred* ook, omdat *kleinKader* zich bevindt in *grootKader*. *kleinKader* heeft dus ook direct toegang tot *fred*:

```
; gekoppeld aan kleinKader
method mouseExit (var eventInfo MouseEvent)
fred = 21 ; stelt waarde van grootKader-variabele fred in op 21
message (fred)
endMethod
```

*grootKader*, *kleinKader* en *mijnCirkel* hebben dus allemaal direct toegang tot *tc*, die is gedeclareerd in het Var-venster voor de onderliggende pagina, *dePagina*.

*grootKader* kan echter niet *ralph* zien, een variabele die in het Var-venster van *kleinKader* is gedeclareerd, omdat *kleinKader* lager staat in de hiërarchie van ingesloten objecten. *grootKader* kan geen variabelen zien die in *mijnCirkel* zijn gedeclareerd. De methodes in *mijnCirkel*

hebben geen toegang tot variabelen die in *grootKader* en *kleinKader* zijn gedeclareerd.

***Opmerking*** Als u het kenmerk 'Objecten insluiten' van een object deselecteert, wordt dat object door ObjectPAL niet als een insluitend object behandeld. Andere objecten die zich binnen dat object bevinden, hebben geen directe toegang tot de variabelen van het object.

Het formulier vormt het hoogste niveau in de hiërarchie van ingesloten objecten. Variabelen die in het Var-venster van een formulier zijn gedeclareerd, zijn toegankelijk voor alle objecten op het formulier. Stel, uitgaande van Afbeelding 11-11, dat er een methode is gekoppeld aan *kleinKader* die een waarde van 100 toewijst aan de variabele *li* (op het formulier gedeclareerd). Stel vervolgens dat een methode die is gekoppeld aan *mijnCirkel*, de volgende instructie bevat:

```
msgInfo("1 optellen bij variabele li", li + 1)
```

Deze instructie geeft de waarde 101 in een dialoogvenster weer, omdat *mijnCirkel* de variabele *li* kan zien en weet dat de waarde van deze variabele 100 is.

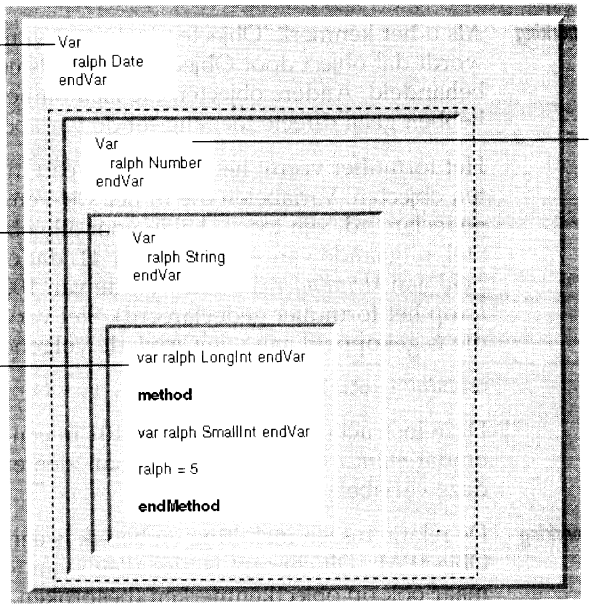
***Opmerking*** De relatie tussen insluiting en bereik is een belangrijk concept in ObjectPAL. Dit concept is niet alleen van toepassing op variabelen, maar ook op objectkenmerken (besproken in Hoofdstuk 13), op eigen methodes en procedures die aan objecten zijn gekoppeld (gedemonstreerd in de online voorbeelden), op de manier waarop objecten acties behandelen en op de manier waarop ObjectPAL fouten behandelt (dit wordt besproken in Hoofdstuk 19).

## Verbinding tijdens compilatie

In programmeertermen is het "verbinden" van een variabele het associëren van een variabele met een gegevenstype. De compiler van ObjectPAL verbindt variabelen tijdens de compilatie van de broncode. ObjectPAL kent geen runtime verbinding. Als de compiler een variabele tegenkomt in een instructie, wordt de rest van de broncode doorzocht om te bepalen waar de variabele is gedeclareerd, zodat de variabele met het gedeclareerde gegevenstype kan worden verbonden.

In Afbeelding 11-12 ziet u waar de compiler naar declararties van variabelen zoekt. Eerst wordt gezocht tussen **method** en **endMethod**, vervolgens boven **method**, daarna in het Var-venster van het object en in het Var-venster van het insluitend object, totdat het formulier wordt bereikt. De compiler werkt met de eerste geschikte declaratie die wordt gevonden. Als er een declaratie is gevonden, wordt niet verder gezocht. Als een variabele niet is gedefinieerd, behandelt de compiler deze als een AnyType-variabele.

Afbeelding 11-12 Verbinding tijdens compilatie




U kunt variabelen op elk niveau in de hiërarchie van ingesloten objecten declareren, maar dat hoeft niet. Als u alleen globale variabelen (zichtbaar voor alle objecten op een formulier) en lokale variabelen (alleen zichtbaar voor het object waarin de variabelen worden gedeclareerd) wilt, kunt u globale variabelen declareren in het Var-venster van het formulier en, waar nodig, lokale variabelen in methodes of procedures.

***Opmerking*** Globale variabelen zijn globaal voor het formulier waarop deze worden gedeclareerd. Een variabele kan slechts voor één formulier als globaal worden gedeclareerd.

## Levensduur van een variabele

De levensduur van de variabelen die u in het Var-venster van een object declareert, komt overeen met die van het object. Als het object uit het formulier wordt verwijderd, zijn de variabelen van het object niet meer toegankelijk. U kunt variabelen niet expliciet vrijgeven.

Variabelen die u in een methode of een procedure declareert, zijn alleen toegankelijk als de methode of procedure wordt uitgevoerd.

## Door de gebruiker gedefinieerde gegevenstypes

Als u 'Type' kiest in het methodevenster van een object, kunt u zelf synoniemen definiëren voor bestaande gegevenstypes. U gebruikt hierbij de volgende structuur:

```
type
   nweTypeNaam = bestaandType
endType
```

Het bereik van de gegevenstypes die u declareert, wordt op dezelfde manier bepaald als dat van variabelen. Als u een nieuw type declareert, wordt er geen nieuw gegevenstype gemaakt. In plaats daarvan wordt een synoniem gemaakt voor een bestaand type. In het volgende voorbeeld wordt het nieuwe gegevenstype 'Salaris' gedeclareerd op basis van het bestaande type SmallInt.

```
type
   Salaris = SmallInt
endType
```

U kunt dit nieuwe type gebruiken om variabelen te declareren. De volgende twee declaraties zijn bijvoorbeeld hetzelfde:

```
var
   loonstrook Salaris
endVar

var
   loonstrook SmallInt
endVar
```

De eerste declaratie is echter gemakkelijker te onthouden en in een grote applicatie misschien beter te onderhouden. Stel dat u een behoorlijke loonsverhoging krijgt, waarvoor SmallInt niet meer toereikend is. In plaats van alle declaraties van *loonstrook* te wijzigen, verandert u alleen de definitie van 'Salaris':

```
type
   Salaris LongInt
endType
```

Een uiterst nuttig type is Record. Een Record van ObjectPAL komt overeen met **record** in Pascal of **struct** in C. Records die in het Type-venster van een object zijn gedefinieerd, staan volledig los van records die zijn geassocieerd met een tabel. ObjectPAL herkent deze Records als een apart type. Zie Hoofdstuk 15 voor meer informatie.

De volgende code declareert een Record als een gegevenstype:

```
type
   WerknemerRec = Record
                     werknemerNaam String
                     afdNummer Number
                     titel  String
                  EndRecord
endType
```

Als u het type hebt gedeclareerd, kunt u andere variabelen declareren van het type WerknemerRec en kunt u waarden aan de velden van het record toewijzen:

```
var
   FrankBorland = WerknemerRec
endVar
```

```
FrankBorland.werknemerNaam = "Frank Borland"
FrankBorland.afdNummer = 43
FrankBorland.titel = "Genie"
```

Het kan ook van pas komen een type voor een array door de gebruiker te laten definiëren:

```
type
   BasisArray Array[1200] String
endType
```

Vervolgens kunt u de typenaam gebruiken om andere arrays van hetzelfde type te declareren:

```
var
   mijnArray, uwArray BasisArray
endVar
```

## Lege variabelen

Als u lege waarden opslaat in ObjectPAL-variabelen, zijn er drie mogelijke toestanden:

- *Blank* (soms Null genoemd) betekent dat de variabele geen waarde heeft. Dit is de enige toestand die wordt ondersteund in Paradox-tabellen. De reeks (" ") wordt als leeg beschouwd.
- *Err* is het resultaat van ongeldige berekeningen (bijvoorbeeld delingen door nul) in een berekend veld. Err heeft voorrang op Blank, dus beide als argumenten naast elkaar bestaan, is het resultaat Err; dat wil zeggen, Err + Blank = Err. Als Err wordt opgeslagen in een Paradox-tabel, vindt er een conversie plaats naar Blank.
- *UnAssigned* is het resultaat als een variabele geen toegewezen waarde heeft. UnAssigned heeft voorrang op Blank, maar niet op Err. UnAssigned-waarden veroorzaken runtime fouten.

Err en UnAssigned zijn hulpmiddelen bij het opsporen van fouten.

Als u wilt weten of een variabele leeg is, gebruikt u de **isBlank**-methode van het type AnyType. Als u wilt weten of aan een variabele een waarde is toegewezen, gebruikt u de **isAssigned**-methode (alle types).

**isBlank** is niet het tegenovergestelde van **isAssigned**. **isBlank** rapporteert over een toestand van de waarde van een variabele (of deze de waarde "leeg" heeft). **isAssigned** geeft een toestand van een variabele weer (of deze voor gebruik is geïnitialiseerd) en niet de waarde van de variabele. **isBlank** werkt alleen bij variabelen die een toegewezen waarde hebben. In het volgende voorbeeld is de variabele *testMij* noch toegewezen noch leeg:

```
Var testMij SmallInt endVar
message(testMij.isAssigned())          ; geeft False weer
message(testMij.isBlank())             ; geeft False weer
```

De volgende instructies wijzen aan de variabele *testMij* een waarde van 1 toe, waardoor deze niet leeg is, maar wel toegewezen:

```
Var testMij SmallInt endVar
testMij = 1
message(testMij.isAssigned())          ; geeft True weer
message(testMij.isBlank())             ; geeft False weer
```

De volgende instructies kennen expliciet een lege waarde toe aan *testMij*, waardoor deze variabele zowel toegewezen als leeg is:

```
Var testMij SmallInt endVar
testMij = blank()
message(testMij.isAssigned())          ; geeft True weer
message(testMij.isBlank())             ; geeft True weer
```

U kunt de Session-procedure **blankAsZero** gebruiken om lege waarden in berekeningen als nul-waarden te laten behandelen, maar om verwarring te voorkomen is het beter de waarde expliciet in te voeren in plaats van het veld leeg te laten.

Als u met tekenreeksen werkt, behandelt ObjectPAL een lege reeks (" ") als een lege waarde. Bovendien geeft de String-procedure **isSpace** voor een lege waarde True terug. Bijvoorbeeld:

```
var
   testString String
endVar
testString = "" ; wijs de lege reeks toe aan testString

message(testString.isAssigned()) ; geeft True weer
message(testString.isBlank())    ; geeft True weer
message(testString.isSpace())    ; geeft True weer
```

# Constanten definiëren

*U kunt constanten voor één methode definiëren of een Const-venster openen om constanten te definiëren voor alle methodes van een object.*

Constanten lijken op variabelen, maar kunnen niet worden veranderd als het programma wordt uitgevoerd, waardoor de compiler efficiëntere code kan genereren. Het is verstandig symbolische namen te geven aan getallen en reeksen, zodat de betekenis van een constante duidelijk is. Zo kunt u een waarde veranderen door één definitie te wijzigen en hoeft u de constanten niet één voor één op te zoeken.

Als u bijvoorbeeld de constanten *mijnMAX*, *mijnMIN* en *groet* in het Const-venster van een object hebt gedeclareerd, kunt u ernaar verwijzen in elke methode die aan dat object is gekoppeld en in methodes die zijn gekoppeld aan objecten die zich in dat object bevinden.

```
const
   mijnMAX = SmallInt(25)
   mijnMIN = SmallInt(10)
   groet = "Tot verkiekemis."
endConst
```

Als de grenzen van minimale en maximale waarden veranderen of als u liever een andere afscheidgroet wilt, kunt u deze veranderingen allemaal in het Const-venster aanbrengen.

Reeksconstanten worden automatisch in hulpbronnen geplaatst. Hier kunnen deze constanten worden gewijzigd (met een hulpmiddel zoals de Resource Workshop van Borland), zonder dat dit invloed heeft op de broncode.

De structuur voor de declaratie van een constante luidt:

```
const
   constNaam = value
   ; of
   constNaam = typeNaam (value)
   ; om het type om te zetten
endConst
```

Als u een constante declareert, leidt Paradox het gegevenstype voor de constante af van de waarde en wordt de constante beschouwd als een Number, een SmallInt of een String. Als u wilt dat een constante een ander gegevenstype krijgt, moet u deze expliciet *omzetten* (toewijzen), zoals in het volgende voorbeeld.

```
const
   x = 123.45              ; Number, afgeleid
   a = -1000               ; SmallInt, afgeleid
   s = "11-Nov-92"         ; String, afgeleid

   c = Currency(123.45)    ; Geld, omgezet
   n = Number(-1000)       ; Number, omgezet
   d = Date("11/11/92")    ; Date, omgezet
endConst
```

## ObjectPAL-constanten

De ObjectPAL-taal bevat vele voorgedefinieerde constanten. 'Blue' is bijvoorbeeld een constante met de waarde 16711680 die de kleur blauw aanduidt. De volgende instructies zijn identiek:

```
datKader.color = Blue

datKader.color = 16711680
```

Het is echter gemakkelijker een woord te onthouden dan een getal.

Er zijn in ObjectPAL, behalve voor kleuren, ook constanten voor vele andere doeleinden. Informatie over deze constanten vindt u op de volgende plaatsen:

- In de online ObjectPAL Help.
- In een venster van de ObjectPAL-Editor van Paradox. Kies 'Taal | Constanten'.
- In een tabel die u maakt met de System-methode **enumRTLConstants**.

# Argumenten doorgeven

ObjectPAL ondersteunt de volgende conventies voor het doorgeven van argumenten aan methodes en procedures:

- Als verwijzing, dat wil zeggen, als verwijzing naar de oorspronkelijke waarde. De waarde van een argument die u als verwijzing doorgeeft, kan in de aangeroepen methode of procedure worden gewijzigd. Gebruik het sleutelwoord VAR, gevolgd door de naam van het argument en het gegevenstype. Bijvoorbeeld: `VAR mijnGetal Number`.
- Als waarde, een kopie van de oorspronkelijke waarde. (De oorspronkelijke waarde kan niet worden veranderd door de aangeroepen methode of procedure, maar de kopie, de doorgegeven waarde, kan wel worden veranderd.) Gebruik de naam van het argument, gevolgd door het gegevenstype. Bijvoorbeeld:

  ```
  hetWoord String
  ```

  U kunt DDE-, Database-, Query-, Session-, Table- en TCursor-variabelen niet als waarde doorgeven.
- Als een constante die niet kan worden gewijzigd. (De compiler staat geen wijziging toe.) Bij doorgeven als constante wordt een verwijzing naar de oorspronkelijke waarde doorgegeven. Gebruik het sleutelwoord const, gevolgd door de naam van het argument en het gegevenstype. Bijvoorbeeld:

  ```
  const nietVeranderen Date
  ```

Alle ObjectPAL-types kunnen als verwijzing (var) of als constanten (const) worden doorgegeven. De volgende types kunnen als waarde worden doorgegeven: AnyType, Array, Binary, Currency, Date, DateTime, DynArray, Graphic, Logical, LongInt, Memo, Number, OLE, Point, Record, SmallInt, String, Time en UIObject. Deze types kunnen ook worden teruggegeven door eigen methodes en procedures. Bij andere types kan dat niet.

Een argument doorgeven als verwijzing (var) of als constante (const) kan efficiënter zijn dan een argument doorgeven als waarde, omdat u werkt met verwijzingen naar de waarde in plaats van met een kopie van de waarde zelf. In de volgende paragrafen worden voorbeelden gegeven die de effecten laten zien van doorgeven als verwijzing, als waarde en als constante. In elk voorbeeld wordt de variabele *mijnWaarde* aan de eigen procedure *plusEen* doorgegeven. Als *mijnWaarde* als verwijzing wordt doorgegeven, verandert de procedure *plusEen* de waarde van *mijnWaarde*. Als *mijnWaarde* als waarde of als constante wordt doorgegeven, verandert de waarde van *mijnWaarde* niet.

## Doorgeven als verwijzing

In het volgende voorbeeld wordt het sleutelwoord VAR in een eigen procedure gebruikt (deze technieken werken ook in eigen methodes) om een argument als verwijzing door te geven. Als de **pushButton**-methode wordt uitgevoerd, wordt de variabele *mijnWaarde* gedeclareerd, wordt aan deze variabele de waarde 0 toegewezen en wordt de variabele doorgegeven aan de procedure **plusEen**. De eerste regel van **plusEen** declareert dat deze procedure één argument gebruikt, namelijk *mijnWaarde* van het type SmallInt, en een waarde teruggeeft van het type SmallInt. De procedure telt 1 op bij *mijnWaarde*, roept **view** aan om de nieuwe waarde van *mijnWaarde* in een dialoogvenster weer te geven en geeft de nieuwe waarde van *mijnWaarde* terug aan de **pushButton**-methode. De **pushButton**-methode roept vervolgens **view** aan om de waarde van *mijnWaarde* in een ander dialoogvenster weer te geven.

```
proc plusEen(var mijnWaarde SmallInt)
   mijnWaarde = mijnWaarde + 1
   mijnWaarde.view("proc")       ; geef de waarde van mijnWaarde weer (het is 1)
endProc

method pushButton(var eventInfo Event)
var mijnWaarde SmallInt endVar   ; declareer de variabele
   mijnWaarde = 0                ; wijs een beginwaarde toe
   plusEen(mijnWaarde)           ; roep de procedure aan
   mijnWaarde.view("method")     ; geef de waarde van mijnWaarde weer (het is 1)
endMethod
```

Ingebouwde methodes geven het argument *eventInfo* als verwijzing door, wat betekent dat u de waarden niet alleen kunt gebruiken, maar ook kunt veranderen. Zie Hoofdstuk 12 voor meer informatie over *eventInfo*.

## Doorgeven als waarde

Het volgende voorbeeld lijkt op het eerste, behalve dat het sleutelwoord **var** uit de procedure **plusEen** is weggelaten. *mijnWaarde* wordt daarom als waarde doorgegeven: de waarde van *mijnWaarde* wordt aan de procedure doorgegeven en de procedure verandert die waarde, maar de waarde van de variabele *mijnWaarde* in de methode verandert niet.

```
proc plusEen(mijnWaarde SmallInt)
   mijnWaarde = mijnWaarde + 1
   mijnWaarde.view("proc")        ; in de procedure: mijnWaarde = 1
endProc

method pushButton(var eventInfo Event)
var mijnWaarde SmallInt endVar
   mijnWaarde = 0
   plusEen(mijnWaarde)
   mijnWaarde.view("method")      ; in de methode: mijnWaarde = 0 (is
                                  ; ongewijzigd)
endMethod
```

## Doorgeven als constante

In het volgende voorbeeld wordt het sleutelwoord const in een pro-cedure gebruikt om een waarde als constante door te geven. Als u de syntaxiscontrole op deze methode uitvoert, verschijnt de foutmelding **Fout: Aan constante variabele kan geen waarde worden toegewezen.** Het sleutelwoord **const** in de procedure **plusEen** declareert *mijnWaarde* als constante en een constante waarde kan per definitie niet veranderen.

De volgende instructie veroorzaakt de fout, omdat deze instructie de waarde van *mijnWaarde* probeert te veranderen:

```
mijnWaarde = mijnWaarde + 1
```

De procedure **plusEen** ziet er als volgt uit:

```
proc plusEen(const mijnWaarde SmallInt)
   mijnWaarde = mijnWaarde + 1     ; deze regel veroorzaakt een compilatiefout
   mijnWaarde.view("proc")
endProc

method pushButton(var eventInfo Event)
   var mijnWaarde SmallInt endVar
   mijnWaarde = 0
   plusEen(mijnWaarde)
   mijnWaarde.view("methode")
endMethod
```

In de volgende code wordt de plaatshoudervariabele *houder* gebruikt om de fout te corrigeren. De variabele *houder* wordt boven de procedure en de methode gedeclareerd, zodat deze zichtbaar is voor beide. Nu wordt niet geprobeerd de waarde van *mijnWaarde* te veranderen, maar wordt 1 opgeteld bij *mijnWaarde* en wordt het resultaat opgeslagen in *houder*. Vervolgens wordt *houder* aan de aanroepende methode teruggegeven.

```
var houder SmallInt endVar ; definieer een plaatshoudervariabele
                           ; zichtbaar voor de proc en de methode
proc plusEen(const mijnWaarde SmallInt)
   houder = mijnWaarde + 1
endProc

method pushButton(var eventInfo Event)
   var mijnWaarde SmallInt endVar
   mijnWaarde = 0
   plusEen(mijnWaarde)
   mijnWaarde.view("mijnWaarde")          ; geeft 0 weer
   houder.view("houder")                  ; geeft 1 weer
endMethod
```

DEEL III

# Objecttypes

In dit gedeelte van de handleiding worden de belangrijkste onderdelen van een ObjectPAL-applicatie gedetailleerd besproken.

- In Hoofdstuk 12, "Acties", wordt het ObjectPAL-actiemodel beschreven en worden objecten besproken die acties behandelen. Deze acties kunnen door de gebruiker worden gegenereerd via de gebruikersinterface, en door ObjectPAL.
- In Hoofdstuk 13, "Ontwerpobjecten", worden de objecten besproken die de gebruikersinterface vormen voor een applicatie: UIObject, Menu en PopUpMenu.
- In Hoofdstuk 14, "Weergavebeheer", worden objecten besproken die bepalen hoe gegevens aan de gebruiker worden getoond: Application, Form, Report en TableView.
- In Hoofdstuk 15, "Gegevenstypes", worden de gegevenstypes van ObjectPAL besproken: AnyType, Array, Binary, Currency, Date, DateTime, DynArray, Graphic, Logical, LongInt, Memo, Number, OLE, Point, SmallInt, String en Time.
- In Hoofdstuk 16, "Gegevensmodel-objecten", worden objecten besproken die toegang verschaffen tot en informatie geven over gegevens die zijn opgeslagen in tabellen: Database, Query, Table en TCursor.
- In Hoofdstuk 17, "Systeemgegevens-objecten", worden objecten besproken voor het opslaan en manipuleren van gegevens die niet worden opgeslagen in tabellen: DDE, FileSystem, Library, Session, System en TextStream.

HOOFDSTUK 12

# Acties

In dit hoofdstuk worden de objecttypes besproken die acties behandelen. Deze acties kunnen via de gebruikersinterface worden gegenereerd door de gebruiker of door ObjectPAL. In dit hoofdstuk worden het ObjectPAL-actiemodel, de methodes die alle actietypes gemeen hebben, en de afzonderlijke actietypes besproken. In Tabel 12-1 wordt een overzicht gegeven van de actietypes van ObjectPAL.

Tabel 12-1 Acties

| Type | Beschrijving |
|---|---|
| ActionEvent | Informatie over basisactiviteiten |
| ErrorEvent | Informatie over fouten |
| Event | Informatie over acties in het algemeen |
| KeyEvent | Informatie over toetsenbordacties |
| MenuEvent | Informatie over de interactie van de gebruiker met een menu |
| MouseEvent | Informatie over de muis en muishandelingen |
| MoveEvent | Informatie over de verplaatsing van de aanwijzer tussen objecten |
| StatusEvent | Informatie over berichten die verschijnen op de statusregel |
| TimerEvent | Informatie over acties die met opgegeven intervallen worden gegenereerd |
| ValueEvent | Informatie over veranderingen in een veldwaarde |

Als u communiceert met een ObjectPAL-applicatie, genereert u acties. Voor elke actie maakt Paradox een pakket informatie dat naar het formulier wordt gezonden. Het actiepakket wordt standaard door het formulier bekeken en naar het doelobject gestuurd. Het doelobject voert vervolgens de juiste ingebouwde methode uit. Dit gedrag wordt beheerd door het actiemodel van ObjectPAL. In deze paragraaf wordt het actiemodel beschreven en worden de volgende onderwerpen besproken:

- Acties: een eerste blik

- Interne en externe acties
- Het actiepakket: *eventInfo*
- Methodes die alle actietypes gemeen hebben

# Acties: een eerste blik

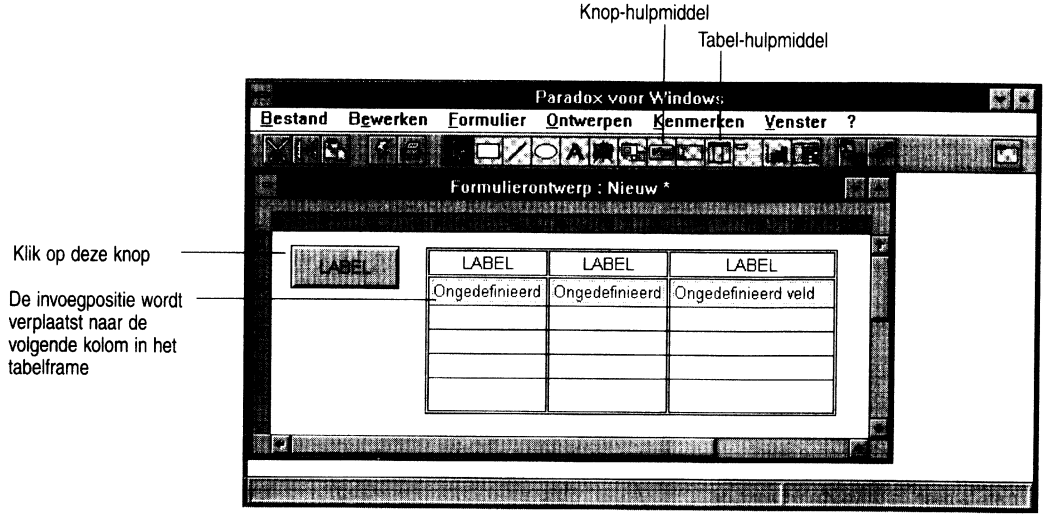
In dit voorbeeld maakt u kennis met het *actiemodel* van ObjectPAL, de regels die Paradox gebruikt om acties te verwerken. Het actiemodel van ObjectPAL is krachtig en geeft een programmeur grote flexibiliteit. Als u het model begrijpt, kunt u de kracht van ObjectPAL maximaal benutten.

Zoals eerder besproken, maakt u Paradox-applicaties door objecten op een formulier te plaatsen en ObjectPAL-code te schrijven die definieert hoe de objecten reageren op acties. Als u de applicatie gebruikt, communiceert u met deze objecten en genereert u acties, waardoor de code wordt uitgevoerd. Eén actie van de gebruiker veroorzaakt een kettingreactie van acties en ingebouwde methodes die standaard worden uitgevoerd. Als u uw eigen code koppelt aan de juiste ingebouwde methodes van de juiste objecten, kunt u precies bepalen wanneer en hoe objecten reageren.

## Formulieren ontwerpen

Het formulier dat u in dit voorbeeld maakt, sluit een knop en een tabelframe in (tabelframes worden besproken in het *Handboek*). U plaatst deze objecten op het formulier en koppelt vervolgens code aan de knop, zodat de invoegpositie naar de volgende kolom in het tabelframe gaat als u op de knop klikt. In Afbeelding 12-1 ziet u het volledige formulier. Onder deze afbeelding volgen de stappen om dit formulier te maken en er code aan te koppelen.

Afbeelding 12-1 Het volledige formulier




1. Kies om te beginnen 'Bestand | Nieuw | Formulier'op het bureaublad en accepteer de standaardwaarden om een leeg formulier te maken.
2. Gebruik het Knop-hulpmiddel om een knop te maken in de linkerbovenhoek van het formulier.
3. Gebruik het Tabel-hulpmiddel om een leeg tabelframe rechts van de knop te maken. Vergelijk uw formulier met Afbeelding 12-1.
4. Noem het tabelframe *TFrame*. Als u een object een naam wilt geven, inspecteert u het en kiest u de naam van het object in het objectmenu. Typ de nieuwe naam in het dialoogvenster.

## Code koppelen

Nu het ontwerp van het formulier voltooid is, bestaat de volgende stap uit het koppelen van code aan de knop, zodat deze code wordt uitgevoerd als u op de knop klikt. Als u uw eigen applicaties maakt, kunt u in elke gewenste volgorde objecten maken en code koppelen.

1. Inspecteer de knop.
2. Kies 'Methodes'.

   *Snelle manier*

   Selecteer de knop en druk op *Ctrl-Spatiebalk* om het methodevenster te openen.

3. Bewerk de **pushButton**-methode als volgt:

```
method pushButton(var eventInfo Event)
   TFrame.action(MoveRight)
endmethod
```

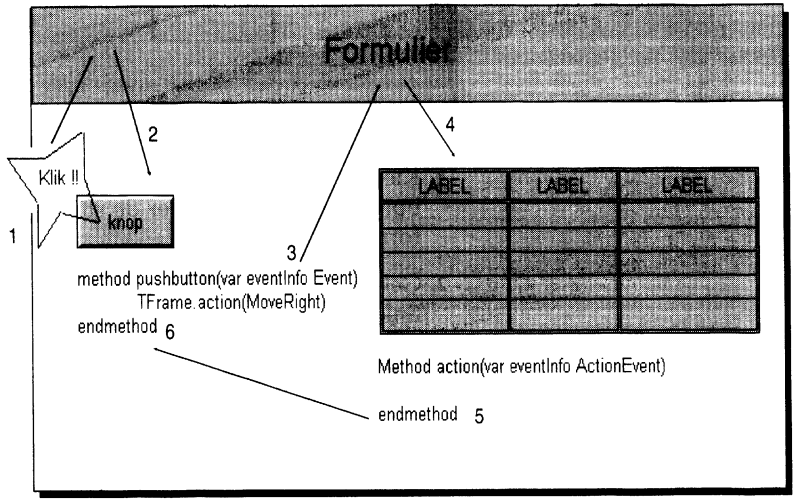
4. Sluit het Editor-venster.
5. Kies 'Formulier | Gegevens tonen'.
6. Klik een paar keer op de knop. De invoegpositie (markering) in het tabelframe gaat steeds verder naar rechts, totdat het meest rechtse veld in het tabelframe wordt bereikt.

## Werking

Als u in dit formulier op een knop klikt, wordt de code uitgevoerd die is gekoppeld aan de knop en verplaatst de invoegpositie zich in het tabelframe. In Afbeelding 12-2 ziet u wat er gebeurt.

Afbeelding 12-2 Een reeks acties en methodes

Elk object in een formulier, inclusief het formulier zelf, heeft ingebouwde methodes die als reactie op acties worden uitgevoerd. De code wordt standaard uitgevoerd, u hoeft hiervoor niets te doen. In dit voorbeeld wordt de invoegpositie verplaatst door de standaardcode van de ingebouwde **action**-methode van het tabelframe.




In Afbeelding 12-2 ziet u een diagram van de acties die worden beschreven in de genummerde stappen hieronder.

1. Eerst klikt u met de muis, waardoor u een actie genereert. Elke actie gaat eerst naar het formulier. Het formulier interpreteert de actie en beslist wat ermee moet gebeuren.
2. Omdat u met de muis hebt geklikt toen de aanwijzer zich boven de knop bevond, wordt de ingebouwde **pushButton**-methode van het formulier uitgevoerd en wordt standaard de ingebouwde **pushButton**-methode van de knop aangeroepen.
3. Vervolgens wordt de **pushButton**-methode van de knop uitgevoerd. De knop lijkt ingedrukt. Als de instructie die u aan de knop hebt gekoppeld, wordt uitgevoerd, genereert deze een andere actie. Deze actie gaat naar het formulier.

```
TFrame.action(MoveRight)
```

4. Het formulier interpreteert deze actie. De ingebouwde **action**-methode van het formulier wordt uitgevoerd en de ingebouwde **action**-methode van het object, genaamd *TFrame*, wordt aangeroepen.
5. De standaardcode van de ingebouwde **action**-methode van *TFrame* wordt uitgevoerd, hoewel u hieraan geen code hebt gekoppeld. De invoegpositie gaat naar rechts.
6. Ten slotte wordt de standaardcode van de ingebouwde **pushButton**-methode van de knop uitgevoerd en de knop springt weer terug. De verwerking van de actie is voltooid.

In Afbeelding 12-2 wordt een eenvoudig overzicht gegeven. In de rest van dit hoofdstuk wordt het model gedetailleerder beschreven.

## Interne en externe acties

ObjectPAL kent twee soorten acties: interne en externe. Interne acties worden gegenereerd binnen Paradox. Voorbeelden van interne acties zijn het openen en sluiten van een object, de aankomst in en het vertrek uit een object en timer-acties. Externe acties worden gegenereerd door de gebruiker (of in een ObjectPAL-methode die de handeling van een gebruiker simuleert). Voorbeelden van externe acties zijn de druk op een toets, een klik met de muis en een keuze in een menu. Tabel 12-2 en Tabel 12-3 geven een overzicht van de ingebouwde methodes die reageren op interne en externe acties.

Elk ontwerpobject heeft ingebouwde standaardmethodes die op acties reageren. In ObjectPAL hebt u de beschikking over de volgende sleutelwoorden om te bepalen wanneer (en of) de standaardcode wordt uitgevoerd: doDefault, disableDefault, enableDefault en passEvent. Zie Appendix B in deze handleiding en de online ObjectPAL Help voor informatie over ingebouwde methodes, standaardcode en sleutelwoorden.

Tabel 12-2 Ingebouwde methodes voor interne acties

| Methode | Beschrijving |
|---|---|
| open | Wordt voor elk object uitgevoerd als het formulier wordt gestart |
| close | Wordt voor elk object uitgevoerd als het formulier wordt gesloten |
| canArrive | Vraagt toestemming om naar een object te gaan |
| canDepart | Vraagt toestemming om een object te verlaten |
| arrive | Wordt uitgevoerd als u naar een object gaat |
| depart | Wordt uitgevoerd als u een object verlaat |

| Methode | Beschrijving |
|---|---|
| setFocus | Wordt uitgevoerd als een object gereed is om invoer vanaf het toetsenbord te ontvangen |
| removeFocus | Wordt uitgevoerd als een object focus verliest |
| timer | Wordt steeds uitgevoerd als een opgegeven tijdsinterval verstrijkt |
| mouseEnter | Wordt uitgevoerd als de aanwijzer binnen de grenzen van een object komt |
| mouseExit | Wordt uitgevoerd als de aanwijzer buiten de grenzen van een object komt |
| pushButton* | Wordt uitgevoerd als u op een knopobject klikt |
| newValue** | Wordt uitgevoerd om te melden dat een veldobject een nieuwe waarde heeft |
| changeValue** | Wordt uitgevoerd om veranderingen door te voeren in een veldwaarde |

* Alleen knoppen en afrollijsten

** Alleen veldobjecten

Tabel 12-3 Ingebouwde methodes voor externe acties

| Methode | Beschrijving |
|---|---|
| mouseMove | Wordt uitgevoerd als de muis wordt verplaatst |
| mouseDown | Wordt uitgevoerd als de linkermuisknop wordt ingedrukt |
| mouseUp | Wordt uitgevoerd als de linkermuisknop wordt losgelaten |
| mouseClick | Wordt uitgevoerd als de linkermuisknop wordt ingedrukt en losgelaten terwijl de aanwijzer zich binnen de grenzen van een object bevindt |
| mouseDouble | Wordt uitgevoerd als wordt gedubbelklikt met de linkermuisknop |
| mouseRightDown | Wordt uitgevoerd als de rechtermuisknop wordt ingedrukt |
| mouseRightUp | Wordt uitgevoerd als de rechtermuisknop wordt losgelaten |
| mouseRightDouble | Wordt uitgevoerd als wordt gedubbelklikt met de rechtermuisknop |
| keyPhysical | Wordt uitgevoerd als er op een toets wordt gedrukt |
| keyChar | Wordt uitgevoerd als een toetsaanslag een ANSI-teken aanduidt dat kan worden ingevoegd |
| action | Wordt uitgevoerd als een toetsaanslag een handeling aanduidt |
| menuAction | Wordt uitgevoerd als u een optie uit een menu kiest of klikt op een knop op de TurboBalk die met een menukeuze correspondeert |
| error | Wordt uitgevoerd als ObjectPAL een fout tegenkomt |
| status | Wordt uitgevoerd als er een bericht verschijnt op de statusbalk |

## Hoe Paradox acties behandelt

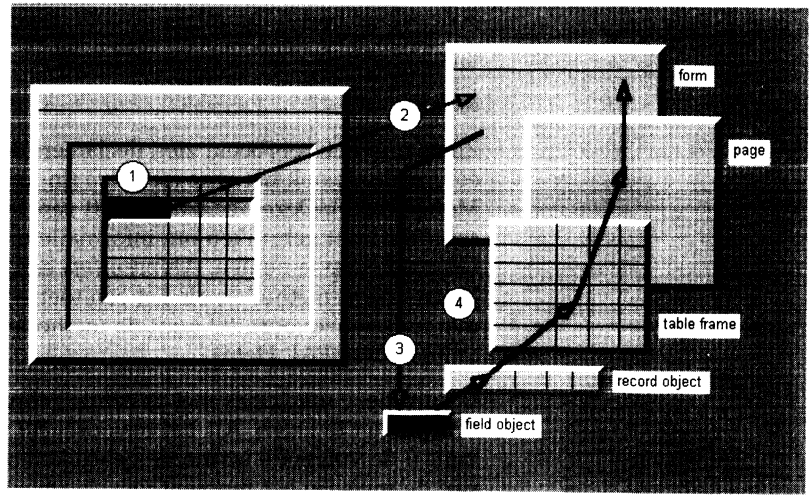
*Acties gaan eerst naar het formulier*

Alle acties, interne en externe, gaan eerst naar het formulier. In ObjectPAL-termen: het formulier *filtert* alle acties. Als het een interne actie is, weet Paradox welk object het doel is. Het formulier zendt het actiepakket *eventInfo* (verderop in dit hoofdstuk gedetailleerder beschreven) dus standaard naar dat object. Het actiepakket activeert de juiste ingebouwde methode.

*Externe acties "borrelen omhoog"*

Bij externe acties gebruikt Paradox echter een mechanisme genaamd *omhoogborrelen* om deze acties in de hiërarchie van ingesloten objecten van object naar object door te geven. Als een object een externe actie ontvangt die geen betekenis heeft (een toetsaanslag heeft bijvoorbeeld geen betekenis voor een ellips), wordt de actie-informatie doorgegeven naar het insluitend object. De informatie borrelt omhoog, zoals een belletje in vloeistof, totdat een object de informatie verwerkt, of totdat de informatie het hoogste niveau, het formulier, bereikt. Het is dus mogelijk dat het formulier dezelfde externe actie twee keer ziet, zoals in Afbeelding 12-3 wordt getoond.

Afbeelding 12-3 Externe acties borrelen




In Afbeelding 12-3 ziet u een diagram van het actieverloop dat ontstaat als u een teken typt of in een veldobject klikt.

1. Klik eerst in het veld of typ een teken in het veldobject.
2. De actie gaat voor de eerste keer naar het formulier. Het formulier weet welk object het doel is en fungeert als verzendstation voor de actie.
3. Het formulier verzendt de actie naar het veldobject en roept de juiste ingebouwde methode aan.

4. Vervolgens kan de actie omhoogborrelen in de hiërarchie van ingesloten objecten (veld, record, tabelframe, pagina, formulier). Terwijl de actie omhoogborrelt, verandert *self*, maar het veld is altijd het doel.

Het formulier filtert alle acties, onderzoekt de acties en zendt deze naar het beoogde doelobject. Van hieruit borrelt de actie van object naar object in de hiërarchie van ingesloten objecten, totdat de actie weer bij het formulier aankomt. Als u bijvoorbeeld tekens in een veldobject typt, is dit veldobject het *beoogde* doel, maar in werkelijkheid gaat de actie eerst naar het formulier. (Zie Afbeelding 12-3.) Als het formulier de actie naar het veldobject verzendt, is het veldobject het werkelijke doel. Het veldobject blijft het doel totdat de actie geheel is verwerkt.

## Het actiepakket: *eventInfo*

Elke actie genereert een pakket informatie over zichzelf en elke ingebouwde methode heeft de parameter *eventInfo*, die het actiepakket bevat. De volgende code is een voorbeeld:

```
method mouseEnter (var eventInfo MouseEvent)
; hoofddeel van de methode komt hier
endMethod
```

Zoals u in het voorbeeld ziet, declareert de standaarddeclaratie van een methode ook het type *eventInfo* (in het voorbeeld is *eventInfo* een MouseEvent). Deze declaratie geeft aan welke methodes u met *eventInfo* kunt gebruiken om informatie uit het pakket te halen en in het pakket op te nemen. Als *eventInfo* een muisactie is, gebruikt u MouseEvent-methodes; als het een toetsactie is, gebruikt u KeyEvent-methodes enzovoort. (U kunt de methodes voor elk type vinden in de online ObjectPAL Help)

U kunt deze methodes met *eventInfo* gebruiken om gegevens uit het actiepakket te halen. Als het bijvoorbeeld een MouseEvent betreft, kunt u de volgende instructie gebruiken om de x- en y-coördinaten van de muis op het moment van de actie te bepalen.

```
eventInfo.getMousePosition(muisX, muisY)
```

Als er sprake is van een KeyEvent, kunt u de volgende instructie gebruiken om te bepalen welk teken is ingedrukt.

```
eventInfo.char()
```

U kunt ook methodes gebruiken om informatie aan het pakket toe te voegen. In het volgende voorbeeld wordt de informatie in het actiepakket die zegt dat er op "a" is gedrukt, vervangen door informatie die zegt dat er op "z" is gedrukt.

```
if eventInfo.char() = "a" then
   eventInfo.setChar("z")
endIf
```

Nog een voorbeeld: in de volgende methode ziet u hoe u het actiepakket gebruikt om op een eenvoudige manier waarden te controleren.

```
; deze methode is gekoppeld aan een niet-verbonden veld
method canDepart(var eventInfo MoveEvent)
   if self.value > 50 then
      eventInfo.setErrorCode(CanNotDepart) ; CanNotDepart is een niet-nul
                                           ; constante
   endIf
endmethod
```

Deze code voorkomt dat de invoegpositie het veld verlaat als dit veld een waarde bevat die groter is dan 50. Anders beslist de ingebouwde veldvalidatie of de waarde geldig is.

***Opmerking*** Het argument *CanNotDepart* is een constante die door ObjectPAL is gedefinieerd voor gebruik met **setErrorCode**. Deze constante heeft per definitie een niet-nul waarde. Constanten kunnen online worden weergegeven. Als u de lijst wilt zien, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten'. Kies vervolgens EventReturns in de kolom 'Types constanten'. De constanten verschijnen in de kolom 'Constanten'.

## ObjectPAL en de behandeling van acties

Omdat elke actie eerst naar het formulier gaat, heeft het formulier de kans om een actie te behandelen voordat deze naar het beoogde doel gaat. In het geval van externe acties kan het formulier dit ook doen nadat de actie het beoogde doel heeft bereikt. Als u code op formulierniveau koppelt aan een actie, is het belangrijk te weten of het formulier zelf een actie behandelt of deze filtert, voordat het de actie doorgeeft aan een ander object. ObjectPAL beschikt over de methode (**isPreFilter**), die op beide vragen antwoord geeft.

### *isPreFilter*

De **isPreFilter**-methode is gedefinieerd voor alle actietypes en beantwoordt twee vragen:

- Ziet het formulier deze actie voor de eerste keer?
- Is het formulier het doel?

**isPreFilter** is zo'n belangrijke methode dat deze voor elk ingebouwde methode op formulierniveau is opgenomen in de standaardtekst in het Editor-venster. Als u bijvoorbeeld voor de eerste keer code koppelt aan de ingebouwde **open**-methode van het formulier, bevat het Editor-venster de volgende code:

```
method open(var eventInfo Event)
if eventInfo.isPreFilter()
   then
       ; deze code wordt uitgevoerd voor elk object op het formulier
```

```
   else
      ; deze code wordt alleen uitgevoerd voor het formulier zelf

endIf
endmethod
```

Deze standaardcode wordt voor het gemak ingevoegd in ingebouwde methodes op formulierniveau. Het is aan te bevelen de standaardcode te gebruiken, maar u kunt de code indien nodig verwijderen. (Als u de code verwijdert, let dan op de waarschuwing aan het einde van deze paragraaf.)

**isPreFilter** geeft True terug in de volgende gevallen:

- Als het doel een ander object is dan het formulier en het formulier deze actie nog niet heeft behandeld
- Bij alle interne acties
- Bij externe acties, als deze voor de eerste keer het formulier bereiken

**isPreFilter** geeft False terug als de externe acties terugborrelen naar het formulier.

Als u een formulier wilt maken dat de effecten van **isPreFilter** demonstreert, gaat u als volgt te werk:

1. Maak een leeg, niet-verbonden formulier.
2. Plaats twee kaders in het formulier.
3. Bewerk de ingebouwde **open**-methode van het formulier als volgt:

```
method open(var eventInfo Event)
if eventInfo.isPreFilter()
   then
   beep()   ; deze code wordt uitgevoerd voor alle objecten op het formulier
   else
            ; deze code wordt alleen voor het formulier uitgevoerd
endIf
endmethod
```

4. Start het formulier. Uw systeem moet driemaal een pieptoon laten horen: eenmaal voor elk kader en eenmaal voor de onderliggende pagina (maar *niet* voor het formulier).
5. Ga terug naar het formulierontwerpvenster en bewerk de ingebouwde **open**-methode van het formulier als volgt:

```
method open(var eventInfo Event)
if eventInfo.isPreFilter()
   then
            ; deze code wordt uitgevoerd voor alle objecten op het formulier
   else
   beep()   ; deze code wordt alleen voor het formulier uitgevoerd

endIf
endmethod
```

6. Start het formulier opnieuw. Uw systeem moet slechts één pieptoon laten horen, namelijk wanneer het formulier zelf wordt geopend.

### *getTarget*

U gebruikt de **getTarget**-methode, die is gedefinieerd voor alle actietypes, om te bepalen welk object het beoogde doel van een actie is. Het formulier weet welk object het beoogde doel is (in de meeste gevallen is dit het actieve object). Daarom kan het formulier de actie op de juiste wijze doorzenden. (Het doel van een actie blijft steeds aanwezig in het actiepakket, ongeacht of de actie omhoogborrelt.) De volgende instructies, gekoppeld aan de ingebouwde **keyChar**-methode van een formulier, geven bijvoorbeeld de naam en het type van het doelobject, ongeacht hoeveel objecten het formulier insluit en welk object het doel is:

```
method keyChar(var eventInfo KeyEvent)
   var
      hetDoel UIObject
   endVar

if eventInfo.isPreFilter()
   then
        ; deze code wordt uitgevoerd voor alle objecten op het formulier
   else
        ; deze code wordt alleen voor het formulier uitgevoerd
        eventInfo.getTarget(hetDoel)
        msgInfo("Doeltype:", hetDoel.class)
        msgInfo("Doelnaam:", hetDoel.name)
endIf
endmethod
```

*Overzicht*

De volgende lijst geeft een overzicht van de informatie die wordt teruggegeven door **isFirstTime**, **isTargetSelf** en **isPreFilter**:

- **isFirstTime**: ziet het formulier de actie voor de eerste keer (voordat het de actie verzendt naar het doelobject)?
- **isTargetSelf**: is het formulier het doel van de actie?
- **isPreFilter**: ziet het formulier de actie voor de eerste keer *en* is een ander object het doel?

### *getObjectHit*

Muisacties verschillen van andere acties, omdat de aanwijzer zich vrij over het scherm kan bewegen en omdat het beoogde doel niet het actieve object hoeft te zijn. Voor muisacties meldt de methode **getObjectHit** welk object het beoogde doel is.

Het volgende voorbeeld is verbonden met de ingebouwde **mouseEnter**-methode voor een formulier. Als de aanwijzer een object binnen gaat, bepaalt deze methode of het een veldobject is. Als dit het geval is, toont deze methode de naam van het veld op de statusbalk. Is dit niet het geval, dan wordt er geen bericht getoond.

```
method mouseEnter (var eventInfo MouseEvent)
   var
      hetDoel UIObject
   endVar
if eventInfo.isPreFilter()
   then
        ; deze code wordt uitgevoerd voor alle objecten op het formulier
        eventInfo.getObjectHit(hetDoel) ; Welk object werd door de muis geraakt?
        if hetDoel.class = "Veld" then  ; Als het een veldobject is,
           message(hetDoel.name)        ; toon de naam.
        endIf
   else
        ; deze code wordt alleen voor het formulier uitgevoerd
endIf
endMethod
```

## Standaardcode verwijderen

Als u de standaardcode wilt verwijderen uit een ingebouwde methode op formulierniveau, let dan goed op. Denk eraan: *elke actie gaat eerst naar het formulier.*

**Let Op**

Als u acties op formulierniveau behandelt, ga dan goed na welk object het doel is en wanneer de uitvoering wordt gestart. Stel dat u wilt dat er een query wordt uitgevoerd als u een bepaald formulier opent. U *kunt* de volgende code koppelen aan de ingebouwde **open**-methode en dan *zou* de query worden uitgevoerd als het formulier wordt geopend. Bijvoorbeeld:

*Doe dit niet!*

```
method open(var eventInfo Event)
   executeQBEFile("nword.qbe")
endMethod
```

De query zou echter ook worden uitgevoerd voor *elke* **open**-methode in *elk* object op het formulier! Als het formulier wordt geopend, wordt namelijk de ingebouwde **open**-methode voor elk object in het formulier uitgevoerd. De actie gaat telkens eerst naar het formulier en de standaardcode voor de ingebouwde **open**-methode verstuurt de actie naar de **open**-methode van het doelobject. In dit voorbeeld wordt de instructie **executeQBEFile** uitgevoerd vóór de standaardcode, zodat de query voor elk object op het formulier één keer wordt uitgevoerd.

De juiste benadering is op **isPreFilter** te testen en code te plaatsen in de **else**-clausule:

*Doe dit!*

```
method open(var eventInfo Event)

   if eventInfo.isPreFilter() then
      ; deze code wordt uitgevoerd voor alle objecten op het formulier
   else
      ; deze code wordt alleen voor het formulier uitgevoerd
      executeQBEFile("nword.qbe")
   endif
endmethod
```

In dit voorbeeld wordt de **executeQBEFile**-instructie slechts één keer uitgevoerd: als het formulier de **open**-methode voor zichzelf uitvoert.

# Event: het elementaire actietype

Event is het elementaire actietype. Dit type omvat alle acties die niet vallen onder andere actietypes. Elke actie heeft ingebouwde methodes. Enkele van deze ingebouwde methodes zijn gemeenschappelijk voor alle actietypes. In de volgende paragraaf worden gemeenschappelijke methodes beschreven en wordt uitgelegd hoe u elke methode met verschillende acties gebruikt.

## Gemeenschappelijke methodes voor alle actietypes

De volgende methodes zijn gemeenschappelijk voor alle actietypes:

- **errorCode** geeft informatie over de status van de foutvlag.
- **getTarget** geeft aan welk object de actie heeft ontvangen.
- **reason** geeft aan waarom een actie heeft plaatsgevonden.
- **setErrorCode** stelt de foutvlag in.
- **setReason** geeft de reden voor een actie.

Raadpleeg de online ObjectPAL Help voor een compleet overzicht van de methodes voor elk actietype.

### *setErrorCode en errorCode*

U gebruikt **setErrorCode** met *eventInfo* om de status van een actie te op te geven. Bijvoorbeeld:

```
method canArrive(var eventInfo MoveEvent)
   eventInfo.setErrorCode(CanNotArrive) ; CanNotArrive is een
                                        ; ObjectPAL-constante
endMethod
```

Door deze methode aan een veld te koppelen, kunt u voorkomen dat een gebruiker de aanwijzer in het veld plaatst. Aan de constante CanNotArrive is door ObjectPAL een niet-nul waarde toegekend en elke foutwaarde die niet-nul is, blokkeert de uitvoering van de ingebouwde code van deze methode.

***Opmerking*** Zie Appendix B voor meer informatie over het blokkeren van de standaardcode voor een ingebouwde methode.

Als ObjectPAL een methode kent voor het instellen van informatie, is er meestal ook een methode voor het opvragen van die informatie. Hiervoor dient **errorCode**: deze methode geeft informatie over de status van een actie. In de meeste gevallen weet u de status al: deze heeft de waarde die u instelt, zoals in het voorbeeld hierboven, of heeft de waarde 0.

***Opmerking*** De methode **errorCode** voor de actietypes houdt geen enkel verband met de **errorCode**-procedure van het System-type of met de TRY...ONFAIL...ENDTRY-structuur (besproken in Hoofdstuk 19).

### getTarget

U gebruikt **getTarget** (beschikbaar voor alle actietypes) om te bepalen welk object de huidige actie heeft ontvangen. Met **getTarget** en een **switch...endSwitch**-structuur kunt u algemene routines maken voor de behandeling van acties. Stel dat een formulier twee knoppen insluit, een om een bestaand bestand te openen en een om een nieuw bestand te maken. U zou een **pushButton**-methode kunnen schrijven voor elke knop, maar dit wordt onpraktisch als er meer knoppen komen. Maak dus een eigen methode die op de juiste wijze reageert, op welke knop de gebruiker ook klikt. Bijvoorbeeld:

```
method doeWat(var eventInfo Event) ; een eigen methode gekoppeld aan het
                                   ; formulier
   var obj UIObject endVar
   eventInfo.getTarget(obj)
   switch
      case obj.name = "toenemen"  : Aantal.value = Aantal.value +1
      case obj.name = "afnemen"   : Aantal.value = Aantal.value -1
   endSwitch
endmethod
```

U kunt gemakkelijk instructies aan de **switch...endSwitch** -structuur toevoegen als u meer knoppen aan het formulier toevoegt. Voeg vervolgens een aanroep van de eigen **doeWat**-methode toe aan de **pushButton**-methode van elke knop.

```
method pushButton(var eventInfo Event)
   doeWat(eventInfo) ; vergeet niet eventInfo door te geven aan de eigen methode
endMethod
```

In dit voorbeeld wordt *eventInfo* als argument doorgegeven aan de eigen methode **doeWat**, omdat het actiepakket informatie bevat over de vraag welk object het doel van de actie was. **doeWat** heeft deze informatie nodig om een beslissing te nemen in het **switch...endSwitch**-blok.

### reason

Als een ingebouwde methode wordt geactiveerd door een Event, een ErrorEvent, een MenuEvent, een MoveEvent of een StatusEvent, kunt u **reason** gebruiken om te bepalen waarom dat gebeurt. De **depart**-methode van een veldobject kan bijvoorbeeld worden aangeroepen omdat de gebruiker de aanwijzer uit het veldobject heeft verplaatst, vanwege een ObjectPAL-instructie, of omdat de applicatie werd afgesloten. Met **reason** en ObjectPAL-constanten kunt u opgeven op welke van deze omstandigheden moet worden gereageerd. Tabel 12-4 geeft een overzicht van de constanten.

Tabel 12-4 Reason-constanten

| Methode | Constante | Beschrijving |
|---|---|---|
| Event-methodes: **newValue** | FieldValue | De waarde is veranderd door schuiven, door een bijwerking in een netwerk, door een ObjectPAL-instructie of doordat een gebruiker de waarde van een veld heeft veranderd. |
| | EditValue | Waarde is opgegeven doordat een keuzeknop is geselecteerd of doordat een optie is gekozen in een lijst. |
| | StartupValue | De waarde is opgegeven toen het formulier werd geopend. |
| ErrorEvent-methodes: **error** | ErrorCritical | Toont een bericht in een dialoogvenster. |
| | ErrorWarning | Toont een bericht in het statusgebied. |
| MenuEvent-methodes: **menuAction** | MenuNormal | Knoppen op de TurboBalk en eigen ObjectPAL-menu's. |
| | MenuControl | Systeemmenu, Vergroot- en Verkleinknop |
| | MenuDesktop | ingebouwde menu's van Paradox (bijvoorbeeld 'Bestand', 'Venster', 'Help'). |
| MoveEvent-methodes: **arrive, depart, canArrive, canDepart** | UserMove | De gebruiker heeft de invoegpositie naar een ander object verplaatst (met de muis of het toetsenbord). |
| | StartUpMove | Het formulier wordt geopend. |
| | ShutDownMove | Het formulier wordt gesloten. |
| | RefreshMove | Een verplaatsing is nodig omdat er een record is ingevoegd, verwijderd of verplaatst. |
| | PalMove | Geactiveerd door een ObjectPAL-instructie. |
| StatusEvent-methode: **status** | ModeWindow1 | Bericht gestuurd naar het meest linkse kleine gebied van de statusbalk. |
| | ModeWindow2 | Bericht gestuurd naar het middelste kleine gebied van de statusbalk. |
| | ModeWindow3 | Bericht gestuurd naar het meest rechtse kleine gebied van de statusbalk. |
| | StatusWindow | Bericht gestuurd naar het statusgebied (het grote gebied) van de statusbalk. |

*reason en MoveEvents*

Hier volgt een voorbeeld waarin **reason** met **depart** wordt gebruikt.

```
method depart(var eventInfo MoveEvent)
   if eventInfo.reason() = UserMove then
      doIets() ; voer eigen methode uit
```

```
   endIf
endMethod
```

In het vorige voorbeeld wordt de eigen methode **doelets** alleen uitgevoerd als **depart** wordt aangeroepen vanwege een handeling van de gebruiker. In andere gevallen wordt de actie normaal behandeld.

*reason en StatusEvents*

Er is nog een verzameling reasons geassocieerd met de ingebouwde **status**-methode. Telkens wanneer er een bericht naar een van de vier berichtgebieden op de statusregel wordt gestuurd (door een ObjectPAL-methode of door Paradox), wordt er een **status**-methode geactiveerd. De reasons geven aan in welk gebied het bericht verschijnt. Als u **status** en de hiermee geassocieerde reasons gebruikt, kunt u elk bericht onderscheppen, veranderen of vervangen, naar een van de drie berichtgebieden sturen, toewijzen aan een variabele en deze ergens anders weergeven, of blokkeren.

```
method status(var eventInfo StatusEvent)
   if eventInfo.reason() = ModeWindow1 then
      eventInfo.setReason(StatusWindow) ; bericht doorsturen naar statusgebied
   endIf
endMethod
```

Zie de paragraaf "StatusEvent: de statusbalk besturen", verderop in dit hoofdstuk, voor meer informatie over het werken met de statusregel.

*reason en ErrorEvents*

De reasons die geassocieerd zijn met **error** geven aan of de fout een kritieke fout is of een waarschuwing. Elk ontwerpobject in een formulier (en het formulier zelf) heeft een ingebouwde **error**-methode. Bij elk ontwerpobject, met uitzondering van het formulier, wordt de fout standaard doorgegeven aan het insluitend object Als een fout omhoogborrelt naar het formulier, doet de ingebouwde **error**-methode van het formulier een van de volgende twee dingen:

- Als het een kritieke fout is, toont **error** een bericht in een dialoogvenster.
- Als het een waarschuwing is, toont **error** een bericht op de statusregel, waardoor de normale StatusEvent-verwerking wordt geactiveerd.

Als u **reason** gebruikt, kunt u dit standaardgedrag echter veranderen. Stel dat u een dialoogvenster wilt weergeven voor elke fout, ongeacht of het een kritieke fout is. U kunt dan de **error**-methode van het formulier als volgt aanpassen:

```
method error(var eventInfo ErrorEvent)
   if eventInfo.reason() = ErrorWarning then
      eventInfo.setReason(ErrorCritical)
   endIf
endMethod
```

Als een fout omhoogborrelt, kan elk object in de hiërarchie van insluitende objecten deze onderscheppen, zodat twee of meer objecten op verschillende manieren op dezelfde fout zouden kunnen reageren.

Zie Hoofdstuk 19 voor meer informatie over de behandeling van fouten.

*reason in combinatie met Events*

De reasons die zijn geassocieerd met de ingebouwde **newValue**-methode geven aan waarom de methode werd geactiveerd. Als u bijvoorbeeld een keuzeknop selecteert om een waarde op te geven en vervolgens het veld verlaat, activeert u **newValue** tweemaal, beide keren om een andere reden: de eerste keer is de reden EditValue, de tweede keer is het FieldValue. Zie de paragraaf"ValueEvent: gewijzigde veldwaarden afhandelen", in dit hoofdstuk, en Appendix B voor meer informatie over het gebruik van **newValue**.

## ActionEvent: tabelbewerking en verplaatsing

ActionEvents worden voornamelijk gegenereerd door bewerkingen en verplaatsingen in een tabel. Als u met ActionEvents werkt, gebruikt u normaal gesproken ook de handelingsconstanten van ObjectPAL. Als u bijvoorbeeld wilt voorkomen dat gebruikers een tabelframe bewerken, kunt u het volgende doen:

```
; dit krijgt voorrang op de ingebouwde action-methode van een tabelframe
method action(var eventInfo ActionEvent)
   ; open een dialoogvenster als de gebruiker wil overschakelen op de
   ; bewerkmodus
   if eventInfo.id() = DataBeginEdit then   ; DataBeginEdit is een constante
      msgStop("Stop", "U kunt dit veld niet bewerken.")
      disableDefault
   else
      enableDefault ; anders, normaal gedrag
   endIf
endMethod
```

De methodes **id** en **setId** gebruiken constanten (zoals DataBeginEdit) om informatie over de handeling te krijgen en te geven. Er is een online lijst met handelingsconstanten beschikbaar. Als u de lijst wilt weergeven, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten'. Kies vervolgens in de kolom 'Types constanten' een element dat begint met Action, zoals ActionDataCommands. De constanten worden dan weergegeven in de kolom 'Constanten'.

*Door de gebruiker gedefinieerde handelingsconstanten*

U kunt ook uw eigen handelingsconstanten definiëren, zolang u deze binnen een bepaald bereik houdt. Omdat dit bereik in toekomstige versies van Paradox kan veranderen, kent ObjectPAL de constanten UserAction en MaxUserAction, die de toegestane minimum- en maximumwaarden vertegenwoordigen.

Stel dat u twee handelingsconstanten wilt definiëren, DezeActie en DieActie. In een Const-venster definieert u de waarden voor uw eigen constanten als volgt:

```
Const
   DezeActie = 1
   DieActie = 2
EndConst
```

Als u een van deze constanten wilt gebruiken, telt u deze op bij UserAction:

```
method action(Var eventInfo ActionEvent)
   if eventInfo.id() = DezeActie + UserAction then
      doeIets()
   endIf
endmethod
```

Doordat u UserAction optelt bij uw eigen constante, bent u zeker van een waarde boven het minimum. Als u de waarde onder het maximum wilt houden, moet u de waarde van MaxUserAction controleren. U kunt dit bijvoorbeeld doen door een **message**-instructie te gebruiken:

```
message(MaxUserAction)
```

In deze versie van Paradox is het verschil tussen UserAction en MaxUserAction 2047. Dit betekent dat UserAction + 2047 de hoogste waarde is die u voor een handelingsconstante kunt gebruiken.

***Belangrijk*** Methodes voor het ActionEvent-type zijn nauw verbonden met de ingebouwde **action**-methode. Zie Hoofdstuk 13 en Appendix B voor meer informatie.

# ErrorEvent: informatie over fouten

Het type ErrorEvent kent methodes die u kunt gebruiken om informatie op te vragen en in te stellen voor verwerking door de **error**-methode die is ingebouwd in elk ontwerpobject (UIObject-type). Net als andere acties, gaat een fout eerst naar het formulier, dat de fout vervolgens verzendt naar het beoogde doelobject (normaal gesproken het object waarvan de code de fout heeft veroorzaakt). Met dit model kunt u een centrale routine maken die fouten behandelt, en deze routine vervolgens koppelen aan het formulier. Zie Hoofdstuk 19 voor meer informatie over fouten en de behandeling van fouten.

*Door de gebruiker gedefinieerde foutconstanten*

U kunt ook uw eigen foutconstanten definiëren, zolang u deze binnen een bepaald bereik houdt. Omdat dit bereik in toekomstige versies van Paradox kan veranderen, kent ObjectPAL de constanten UserError en MaxUserError, die de toegestane minimum- en maximumwaarden vertegenwoordigen.

Stel dat u twee foutconstanten wilt definiëren, DezeFout en DieFout. U definieert de waarden voor uw eigen constanten als volgt in een Const-venster:

```
Const
   DezeFout = 1
   DieFout = 2
EndConst
```

Als u vervolgens een van deze constanten wilt gebruiken, telt u deze op bij UserError:

```
method error(Var eventInfo ErrorEvent)
   if eventInfo.id() = DezeFout + UserError then
      doeIets()
   endIf
endmethod
```

Doordat u UserError optelt bij uw eigen constante, bent u zeker van een waarde boven het minimum. Als u de waarde onder het maximum wilt houden, moet u de waarde van MaxUserError controleren. U kunt dit bijvoorbeeld doen met een **message**-instructie:

```
message(MaxUserError)
```

In deze versie van Paradox is het verschil tussen UserError en MaxUserError 2046. Dit betekent dat UserError + 2046 de hoogste waarde is die u kunt gebruiken voor een foutconstante. (De foutcode 0 betekent "geen fout".)

# KeyEvent: toetsenbordhandelingen

Het type KeyEvent bevat methodes die informatie krijgen en geven over toetsenbordhandelingen, zoals:

- Tekens die naar het programma zijn gezonden: **char, charAnsiCode, vChar, vCharAnsiCode, setChar, setVChar**
- De status van *Alt, Ctrl* en *Shift*: **isAltKeyDown, setAltKeyDown, isControlKeyDown, setControlKeyDown, isShiftKeyDown, setShiftKeyDown**

De volgende code gebruikt de KeyEvent-methode **char** in de ingebouwde **keyChar**-methode om een pop-up menu te laten verschijnen als u **?** typt. Als deze code bijvoorbeeld is gekoppeld aan een veldobject, verschijnt het menu als u er **?** in typt.

```
method keyChar (var eventInfo KeyEvent)
   var
      pop PopUpMenu
   endVar

   if eventInfo.char() = "?" then     ; als ? wordt getypt
      disableDefault
      pop.addText("een")              ; maak pop-up menu
```

```
      pop.addText("twee")
      pop.addText("drie")
      self = pop.show()
   endIf                                  ; anders, normaal gedrag
endMethod
```

***Opmerking*** Bepaalde toetscombinaties kunt u niet ondervangen, omdat deze door Windows worden verwerkt voordat ze Paradox bereiken, zoals in de volgende paragraaf wordt uitgelegd.

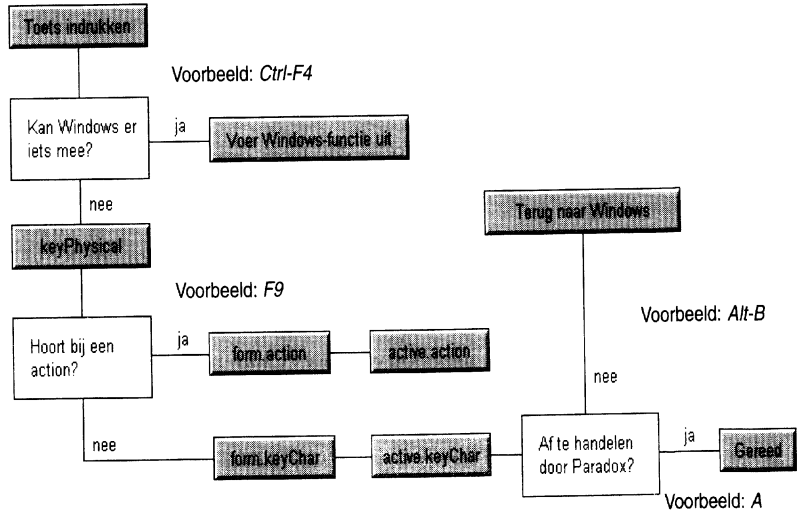
## Toetsenbordacties en ingebouwde methodes

Als u in Paradox op een toets drukt, gebeurt één van de volgende dingen:

- Windows onderschept de toetsaanslag en onderneemt actie.
- Windows geeft de toetsaanslag door aan Paradox en Paradox voert een handeling uit.
- Windows geeft de toetsaanslag door aan Paradox en Paradox geeft deze door aan het actieve object.

Als Windows een toetsaanslag onderschept, voert ObjectPAL geen handeling uit. In de andere twee gevallen kunt u de ingebouwde methodes **keyPhysical**, **action** en **keyChar** gebruiken om te reageren op toetsenbordacties en deze te simuleren. Deze ingebouwde methodes houden nauw verband met, zoals u in Afbeelding 12-4 ziet.

Afbeelding 12-4 Verloop van acties en methodes voor een toetsaanslag




Als u op een toets drukt, ontvangt Windows deze toetsaanslag van de aansturing voor het toetsenbord en wordt de aanslag opgeslagen in de wachtrij voor systeemberichten. Als de toetsaanslag voor Windows

betekenis heeft (bijvoorbeeld *Ctrl-F4*, waardoor het actieve venster wordt gesloten), verricht Windows de bijbehorende handeling. In andere gevallen zendt Windows de toetsaanslag naar Paradox. Paradox genereert een KeyEvent-pakket en stuurt dit naar de **keyPhysical**-methode van het formulier. Als de toetsaanslag overeenkomt met een Paradox-handeling (bijvoorbeeld *F9*, waarmee de bewerkmodus wordt in- en uitgeschakeld), roept Paradox de ingebouwde **action**-methode van het formulier aan met de juiste constante (zoals DataToggleEdit).

Als de toetsaanslag niet door Windows wordt onderschept of vertaald in een handeling door Paradox, geeft de **keyPhysical**-methode van het formulier het actiepakket door aan de **keyChar**-methode, die het pakket standaard doorgeeft aan de **keyChar**-methode van het actieve object. Het actieve object behandelt de toetsaanslag of laat deze omhoogborrelen naar het insluitend object en verder door de hiërarchie van insluitende objecten naar het formulier. Als de toetsaanslag een menusneltoets is (bijvoorbeeld *Alt-B*, waarmee het menu 'Bestand' wordt geopend), geeft het formulier de toetsaanslag voor verwerking terug aan Windows.

***Opmerking*** In dit model wordt alleen de verwerking van toetsaanslagen binnen Paradox beschreven. U kunt geen tekens in andere applicaties verzenden of onderscheppen.

In het volgende eenvoudige voorbeeld wordt getoond hoe dit model werkt.

1. Maak eerst een leeg, niet-verbonden formulier.
2. Plaats een niet-verbonden veldobject op de pagina.
3. Koppel de volgende code aan de ingebouwde **keyPhysical**-methode van het formulier. De code opent een dialoogvenster waarin u wordt verteld op welke toets is gedrukt.

```
method keyPhysical(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ;deze code wordt uitgevoerd voor alle objecten op het formulier
     msgInfo("keyPhysical", eventInfo.vChar())
  else
    ;deze code wordt alleen voor het formulier uitgevoerd
  endif
endmethod
```

4. Koppel de volgende code aan de ingebouwde **keyChar**-methode van het formulier:

```
method keyChar(var eventInfo KeyEvent)
if eventInfo.isPreFilter()
  then
    ;deze code wordt uitgevoerd voor alle objecten op het formulier
     msgInfo("keyChar", eventInfo.vChar())
  else
```

```
    ;deze code wordt alleen voor het formulier uitgevoerd
  endif
endmethod
```

5. Start het formulier.
6. Druk op *Ctrl-F4*. Deze toetsaanslag wordt door Windows onderschept en er verschijnt een dialoogvenster met de vraag of u dit formulier wilt bewaren voordat u het sluit. Kies 'Annuleren' om het dialoogvenster te sluiten. De ObjectPAL-code die u hebt gekoppeld aan de **keyPhysical**- en de **keyChar**-methodes van het formulier, wordt niet uitgevoerd.
7. Druk op *F9*. Windows geeft deze toetsaanslag door aan Paradox. Paradox zendt de toetsaanslag naar de **keyPhysical**-methode van het formulier. De **keyChar**-methode wordt niet uitgevoerd.
8. Druk op *A*. Zowel **keyPhysical** als **keyChar** worden uitgevoerd en het veldobject toont de letter.
9. Druk op *Alt-B*. Zowel **keyPhysical** als **keyChar** worden uitgevoerd en het menu 'Bestand' verschijnt.

De code in het volgende voorbeeld is gekoppeld aan de **keyPhysical**-methode van een niet-verbonden veldobject. Deze verwerkt de meeste toetsaanslagen normaal. Als u bijvoorbeeld de letter **A** typt, wordt er een *A* in het veld geplaatst. Als de gebruiker echter op een pijltoets drukt, verplaatst het veldobject zich! (Dit is gemakkelijker te zien als u het veldobject een andere kleur geeft dan de pagina.) In dit voorbeeld worden virtuele sleutelcodes gebruikt, die in de volgende paragraaf worden uitgelegd.

***Opmerking*** Voor ObjectPAL is het scherm een tweedimensionaal raster, met de oorsprong (0, 0) in de linkerbovenhoek van het insluitend object. De positieve x-waarden nemen naar rechts toe en de positieve y-waarden nemen naar beneden toe.

```
method keyPhysical(var eventInfo KeyEvent)
   var
      x, y LongInt
      posPt Point
   endVar

   disableDefault                    ; voorkom uitvoering van ingebouwde code

   posPt = self.position             ; positie is een kenmerk
   x = posPt.x()
   y = posPt.y()

   switch
      case eventInfo.vCharAnsiCode() = VK_LEFT  : x = x - 100
      case eventInfo.vCharAnsiCode() = VK_RIGHT : x = x + 100
      case eventInfo.vCharAnsiCode() = VK_UP    : y = y - 100
      case eventInfo.vCharAnsiCode() = VK_DOWN  : y = y + 100
      otherwise : enableDefault      ; activeer ingebouwde code
   endSwitch
```

```
   self.position = Point(x, y)
endmethod
```

Let op het gebruik van **disableDefault** en **enableDefault** in het voorbeeld. De aanroep van **disableDefault** aan het begin van de methode voorkomt dat de ingebouwde code door een willekeurige toetsaanslag wordt uitgevoerd. De **switch...endSwitch**-structuur kijkt naar de binnenkomende toetsaanslag. Als het een aanslag van de pijltoetsen is, wordt de waarde van *x* of *y* op de juiste wijze ingesteld en wordt de ingebouwde code niet uitgevoerd. Als de binnenkomende toets geen pijltoets is, roept de OTHERWISE- clausule **enableDefault** aan, zodat de ingebouwde code wordt uitgevoerd en de toetsaanslag normaal wordt verwerkt en in het veldobject wordt weergegeven.

***Opmerking*** Zie de online ObjectPAL Help voor meer informatie over **switch... endSwitch, disableDefault** en **enableDefault**.

In het volgende voorbeeld ziet u hoe u *F1* onderschept. <F50MIF1 roept standaard het Helpsysteem van Paradox aan. Als u uw eigen applicatie maakt, wilt u misschien ook voor een eigen Helpsysteem zorgen en de gebruiker de mogelijkheid geven dit Helpsysteem op de normale wijze aan te roepen. In dit voorbeeld wordt verondersteld dat u een eigen Helpbestand met de naam APPHELP.HLP hebt gemaakt en dat dit bestand zich in uw pad bevindt. Koppel de volgende code aan de ingebouwde **keyPhysical**-methode van het formulier.

```
method keyPhysical(var eventInfo KeyEvent)
   if eventInfo.isPreFilter()
      then
         ; deze code wordt uitgevoerd voor alle objecten op het formulier
      else
         ; deze code wordt alleen voor het formulier uitgevoerd
         if eventInfo.vChar() = "VK_F1" then        ; als de gebruiker drukt op
                                                    ; F1
            helpShowContext("appHelp.hlp", appHelp) ; roep Help aan
            disableDefault                          ; voorkom standaardgedrag
         endIf
   endIf
endmethod
```

### Toetsnamen en ANSI-codes

De methode **vChar** voor het KeyEvent-type geeft de virtuele toetsnaam terug als een reeks. Virtuele toetscodes zijn constanten die de standaardtoetsen van een computer vertegenwoordigen, zoals de letter *A* of de *Esc*-toets. Omdat verschillende merken computers verschillende toetsen kunnen hebben, worden de virtuele toetscodes gebruikt voor de verwerking van toetsenbordinvoer. Zo kan één applicatie op een groot aantal verschillende computers worden gebruikt.

Als u bijvoorbeeld op *Return* drukt, geeft **vChar** de reeks "VK_RETURN" terug. De **keyChar**-methode voor het UIObject-type

gebruikt echter een reeks als argument. De methode accepteert geen numerfieke waarde en interpreteert de reeks niet. Dit betekent dat de methode de reeks "and VK_RETURN" niet vertaalt naar de tekens "and" en de ANSI-code voor VK_RETURN. **keyChar** herkent VK_RETURN niet als een virtuele toets, maar als de reeks "VK_RETURN". Als u deze methodes tegelijkertijd wilt gebruiken, moet u enkele conversies uitvoeren. In Tabel 12-5 wordt een overzicht gegeven van de procedures die ObjectPAL voor dit doel kent (deze procedures zijn gedefinieerd voor het String-type).

Tabel 12-5 Procedures voor de conversie tussen ANSI-codes en toetsnamen

| Methode | Beschrijving |
|---|---|
| ChrToKeyName | Geeft de toetsnaam terug van het teken dat zich in een reeks bevindt. |
| VKCodeToKeyName | Geeft de toetsnaam terug die correspondeert met een opgegeven ANSI-code. |
| KeyNameToChr | Geeft een reeks terug met een lengte van 1. |
| KeyNameToVKCode | Geeft een SmallInt terug die de ANSI-code voor de opgegeven toetsnaam vertegenwoordigt. Dit heeft slechts beperkt nut. Als u deze procedure bij algemene programmering gebruikt. |

Hier volgen enkele voorbeelden voor de conversie tussen ANSI-codes en toetsnamen.

```
ChrToKeyName(Chr(VK_CANCEL)) = "VK_CANCEL"
ChrToKeyName(eventInfo.vChar())) = "VK_CANCEL"

VKCodeToKeyName(VK_CANCEL) = "VK_CANCEL"
VKCodeToKeyName(eventInfo.vCharCode()) = "VK_CANCEL"

KeyNameToChr("VK_CANCEL") = Chr(VK_CANCEL)
KeyNameToChr("VK_CANCEL") = eventInfo.vChar()

KeyNameToVKCode("VK_CANCEL") = VK_CANCEL
KeyNameToVKCode("VK_CANCEL") = eventInfo.vCharCode()
```

# MenuEvent: menukeuzes

Het MenuEvent-type bevat methodes voor menu's op de menubalk van de applicatie, inclusief de systeemmenu's van Windows. Als de gebruiker een optie uit een menu kiest (of klikt op een knop op de TurboBalk die een menu-handeling uitvoert), wordt de **menuAction**-methode geactiveerd. Als u de code koppelt aan de **menuAction**-methode van een object, kunt u definiëren hoe het object reageert op keuzes uit eigen menu's die u zelf maakt, op ingebouwde menu's van Paradox en op de systeemmenu's van Windows.

***Belangrijk*** Zie Hoofdstuk 13 voor meer voorbeelden van het werken met menu's.

## Eigen menu's

Hier volgt een eenvoudig voorbeeld dat de menukeuze in een niet-verbonden veld weergeeft als u op een knop klikt.

De volgende code is gekoppeld aan de **pushButton**-methode van een knop. Deze code maakt een eigen menu met drie opties en geeft deze horizontaal weer op de menubalk van de applicatie.

```
method pushButton(var eventInfo Event)
   var
      m Menu
   endVar
   m.addText("een")
   m.addText("twee")
   m.addText("drie")
   m.show()
endMethod
```

De volgende code is gekoppeld aan een niet-verbonden veld. De code toont de menukeuze van de gebruiker in het veld.

```
method menuAction(var eventInfo MenuEvent)
   self = eventInfo.menuChoice() ; geeft de menukeuze weer in het veld
endmethod
```

## Ingebouwde menu's

Paradox kent een identificatienummer toe aan elke optie in de ingebouwde menu's. Met de **id**-methode die is gedefinieerd voor het MenuEvent-type, kunt u testen op keuzes die in ingebouwde menu's worden gemaakt.

Stel dat u een bericht wilt tonen aan de gebruiker als deze uw applicatie afsluit. Als u ingebouwde menu's van Paradox's gebruikt, kunt u de **menuAction**-methode van een object als volgt aanpassen:

```
method menuAction(var eventInfo MenuEvent)
   if eventInfo.id() = MenuFileExit then ; MenuFileExit is een constante
      msgInfo("Tot ziens", "Dank u.")
   endIf
endMethod
```

In dit voorbeeld is MenuFileExit een menu-opdrachtconstante die door ObjectPAL is gedefinieerd om de waarde te vertegenwoordigen die wordt teruggegeven als u in het ingebouwde menu van Paradox de optie 'Bestand | Afsluiten' kiest.

*Door de gebruiker gedefinieerde menuconstanten*

U kunt ook uw eigen menuconstanten definiëren, zolang u deze binnen een bepaald bereik houdt. Omdat dit bereik in toekomstige versies van Paradox kan veranderen, kent ObjectPAL de constanten UserMenu en MaxUserMenu, die de toegestane minimum- en maximumwaarden vertegenwoordigen.

Stel dat u twee menuconstanten wilt definiëren, DezeMenuOptie en DieMenuOptie. U definieert de waarden voor uw eigen constanten dan als volgt in een Const-venster:

```
Const
   DezeMenuOptie = 1
```

```
   DieMenuOptie = 2
EndConst
```

Als u vervolgens een van deze constanten wilt gebruiken, telt u deze op bij UserMenu:

```
method action(Var eventInfo MenuEvent)
   if eventInfo.id() = DezeMenuOptie + UserMenuItem then
      doeIets()
   endIf
endmethod
```

Doordat u UserMenu hebt opgeteld bij uw eigen constanten, bent u zeker van een waarde boven het minimum. Als u de waarde onder het maximum wilt houden, moet u de waarde van MaxUserMenu controleren. U kunt dit bijvoorbeeld doen met een **message**-instructie:

```
message(MaxUserMenu)
```

In deze versie van Paradox is het verschil tussen UserMenu en MaxUserMenu 2047. Dit betekent dat UserMenu + 2047 de hoogste waarde is die u kunt gebruiken voor een menuconstante.

### Systeemmenu's

U kunt **id** en menu-opdrachtconstanten in een ingebouwde **menuAction**-methode van een object gebruiken om te testen op keuzes in een systeemmenu van Windows (zoals 'Pictogram' en 'Maximumvenster'). De volgende code voorkomt bijvoorbeeld dat u het huidige formulier tot een pictogram verkleint:

```
method menuAction(var eventInfo MenuEvent)
   if eventInfo.id() = MenuControlMinimize then
                              ; MenuControlMinimize is een constante
      disableDefault ; voer de standaardcode niet uit
      beep()
      message("Kan dit formulier niet verkleinen.")
   endIf
endMethod
```

## MouseEvent: muishandelingen verwerken

Methodes van het MouseEvent-type geven antwoord op vragen over muishandelingen, zoals de vragen uit Tabel 12-6.

Tabel 12-6 MouseEvent-methodes

| Vraag | Methodes |
|---|---|
| Op welke knop wordt geklikt? | isLeftDown, isMiddleDown, isRightDown, setLeftDown, setMiddleDown, setRightDown |
| Waar is de aanwijzer? | getMousePosition, x, y, setX, setY, isInside, setMousePosition |

| | |
|---|---|
| Wordt er een toets ingedrukt? | isAltKeyDown, isControlKeyDown, isShiftKeyDown, setAltKeyDown, setControlKeyDown, setShiftKeyDown |
| Welk object heeft de actie ontvangen? | getObjectHit, getTarget |

### Reageren op muishandelingen

UIObjecten beschikken over ingebouwde methodes die u kunt aanpassen om op veel normale muishandelingen te reageren. Bovendien kunt u **hasMouse** gebruiken om te ontdekken of een muiswijzer zich boven een object bevindt. U kunt ook **wasLastClicked** en **wasLastRightClicked** gebruiken om te ontdekken of een object het laatste object was dat een muisklik (of een klik met de rechtermuisknop heeft ontvangen).

```
var
   mijnObject UIObject
endVar
mijnObject.attach(mijnTabelframe) ; veronderstel dat een formulier een
                                  ; TabelFrame insluit
if mijnObject.hasMouse() then
   message("Help! Een muis!")
else
   message("Hier is geen muis.")
endIf
if mijnObject.wasLastClicked() then
   message("Ik heb de laatste klik met de linkermuisknop ontvangen.")
endIf
if mijnObject.wasLastRightClicked() then
   message("Ik heb de laatste klik met de rechtermuisknop ontvangen.")
endIf
```

## MoveEvent: verplaatsen tussen objecten

Met methodes voor het type MoveEvent kunt u informatie opvragen en instellen over de acties die optreden als u in een formulier van het ene object naar het andere gaat. De volgende ingebouwde methodes worden door MoveEvents geactiveerd: **arrive, canArrive, canDepart** en **depart**. Deze methodes worden met de andere ingebouwde methodes besproken in Appendix B. In deze paragraaf wordt een overzicht gegeven van de manieren waarop u methodes van het type MoveEvent kunt gebruiken.

### Veldwaarden controleren

Een veelvoorkomende taak is het inspecteren van het actiepakket van MoveEvent voor een eenvoudige waardencontrole. Dit soort code kunt u het best koppelen aan de ingebouwde methodes **canArrive** en **canDepart**, omdat deze methodes toestemming vragen om de verplaatsing uit te voeren, zodat u de kans krijgt waarden te inspecteren en beslissingen te nemen. De ingebouwde methodes

**arrive** en **depart** worden daarentegen uitgevoerd nadat de verplaatsing is uitgevoerd.

Stel dat de volgende code is gekoppeld aan de ingebouwde **canDepart**-methode van een veldobject. Als u probeert de aanwijzer uit dit veldobject weg te halen, wordt deze code uitgevoerd. De code controleert de huidige waarde en als deze hoger is dan 50, verschijnt er een dialoogvenster en wordt er een foutcode ingesteld om te voorkomen dat de invoegpositie het veld verlaat. (Een foutcode die niet nul is, geeft een fout aan).

```
method canDepart(var eventInfo MoveEvent)
   if self.value > 50 then
      msgStop("Stop", "Typ een waarde lager dan 50.")
      eventInfo.setErrorCode(CanNotDepart) ; CanNotDepart is een niet-nul
                                           ; constante
   endIf
endmethod
```

Deze code voorkomt dat de aanwijzer het veld verlaat als het veld een waarde bevat die hoger is dan 50. Als dit niet het geval is, beslist de standaardcode of de waarde geldig is. U kunt een soortgelijke methode gebruiken om een veldwaarde te controleren voordat de invoegpositie ernaartoe gaat. De volgende code opent bijvoorbeeld een dialoogvenster en voorkomt dat de invoegpositie naar het veld gaat als de waarde lager dan 100 is:

```
method canArrive (var eventInfo MoveEvent)
   if self.value < 100 then
      msgStop("Waarde is te laag.", "Druk op de knop TOENEMEN.")
      eventInfo.setErrorCode(CanNotArrive)
   endIf
endMethod
```

Veronderstel in het volgende voorbeeld dat een formulier een multi-record object insluit dat is verbonden met de tabel *Order*, en een knop met de naam *laatsteOngedaan*. Als in een veld een verandering wordt aangebracht, zoeken de methodes **canDepart** en **changeValue** op het formulier de naam en de oorspronkelijke waarde van het veld op. Als een verandering "ongedaan" kan worden gemaakt, verandert de kleur van het label (een tekstvak met de naam *LaatsteLabelOngedaan*) op *laatsteOngedaan* in zwart. Als de verandering niet ongedaan kan worden gemaakt, wordt het label lichtgekleurd (donkergrijze letters op een lichtgrijze knop). Als de gebruiker op de knop *laatsteOngedaan* klikt, kijkt de **pushButton**-methode van de knop of het eigen label lichtgekleurd is. Is dit niet zo, dan maakt de methode de laatste verandering in het veld ongedaan. De volgende code wordt gekoppeld aan het Var-venster voor het formulier:

```
; ditFormulier::Var
Var
  doelObj,                      ; geeft handle voor huidige object
  laatsteDoelVeld       UIObject  ; zoekt het laatst veranderde veld
  laatsteWaarde         AnyType   ; bevat de laatste veldwaarde
endVar
```

Deze code wordt gekoppeld aan de **canDepart**-methode voor het formulier:

```
; ditFormulier::canDepart
method canDepart(var eventInfo MoveEvent)
if eventInfo.isPreFilter()
  then
    ;deze code wordt uitgevoerd voor alle objecten op het formulier
    ; deze code zoekt de inhoud en de veldnaam van het
    ; laatst veranderde veld in hetzelfde record
    eventInfo.getTarget(doelObj)                    ; doelObj gedeclareerd in
                                                    ; Var-venster
    if doelObj.class = "field" then
      if doelObj.Touched = True then
        ; code in changeValue-methode bevat laatste veldinhoud
        laatsteDoelVeld.attach(doelObj)             ; sla handle voor veld op
LaatsteLabelOngedaan.Font.Color = Black             ; maak knoplabel Zwart
      endif
    else    ; indien verplaatsing naar nieuw record: ongedaan maken wordt
            ; lichtgekleurd
      LaatsteLabelOngedaan.Font.Color = DarkGray    ; grijs knoplabel
    endif
  else
    ;deze code wordt alleen voor het formulier uitgevoerd

endif
endmethod
```

Deze code wordt gekoppeld aan de **changeValue**-methode van het formulier:

```
; ditFormulier::changeValue
method changeValue(var eventInfo ValueEvent)
if eventInfo.isPreFilter()
  then
    ;deze code wordt uitgevoerd voor alle objecten op het formulier
    ; deze code zoekt de oorspronkelijke waarde van het veld dat gaat veranderen
    if doelObj.Touched = True then
      laatsteWaarde = String(doelObj.value)          ; doelWaarde opslaan
    endif
  else
    ;deze code wordt alleen voor het formulier uitgevoerd

endif
endmethod
```

Dit is de code voor de **pushButton**-methode van *laatsteOngedaan*:

```
; laatsteOngedaan::pushButton
method pushButton(var eventInfo Event)
if LaatsteLabelOngedaan.Font.Color = DarkGray then
  ; als knop lichtgekleurd is, is er voor dit record geen verandering die
  ; ongedaan moet worden gemaakt
else  ; als knop niet lichtgekleurd is: doorgaan en vorige veld veranderen
x  msgInfo("Status", "Waarde van " +
                     laatsteDoelVeld.name + " veranderen in " +
                     laatsteWaarde)          ; gebruiker vertellen wat er gebeurt
  laatsteDoelVeld.Value = laatsteWaarde      ; oorspronkelijke waarde herstellen
  laatsteOngedaanLabel.Font.Color = DarkGray  ; waarde kan niet tweemaal
                                              ; ongedaan worden gemaakt!
endif
endmethod
```

Zie de paragraaf "ValueEvent: gewijzigde veldwaarden afhandelen", verderop in dit hoofdstuk, voor meer informatie over het controleren van waarden.

### Foutcodes instellen

U gebruikt **setErrorCode** samen met *eventInfo* om de status van een MoveEvent op te geven. Bijvoorbeeld:

```
method canArrive(var eventInfo MoveEvent)
   eventInfo.setErrorCode(CanNotArrive) ; CanNotArrive is een
                                        ; ObjectPAL-constante
endMethod
```

Als u de bovenstaande methode aan een veld koppelt, voorkomt u dat de gebruiker de aanwijzer in het veld plaatst. Aan de constante CanNotArrive is door ObjectPAL een niet-nul waarde toegekend en elke niet-nul waarde blokkeert de uitvoering van de ingebouwde code van deze methode. In plaats hiervan probeert Paradox naar het volgende object in de tabvolgorde te gaan.

***Opmerking*** De aanroep van **setErrorCode** activeert het mechanisme voor foutbehandeling van Paradox (beschreven in Hoofdstuk 19) niet. Deze aanroep stelt alleen informatie in die door de standaardcode wordt gebruikt.

### reason en MoveEvents

U kunt de **reason**-methode van het MoveEvent-type gebruiken om te bepalen waarom een verplaatsing wordt uitgevoerd. Hier volgt een voorbeeld waarin **reason** met **canDepart** wordt gebruikt:

```
method canDepart(var eventInfo MoveEvent)
   if eventInfo.reason() = UserMove then
      doIets() ; voer eigen methode uit
   endIf
endMethod
```

In dit voorbeeld wordt de eigen methode **doIets** alleen uitgevoerd als **canDepart** wordt aangeroepen door een handeling van de gebruiker; anders wordt de actie normaal behandeld. (UserMove is een ObjectPAL-constante.)

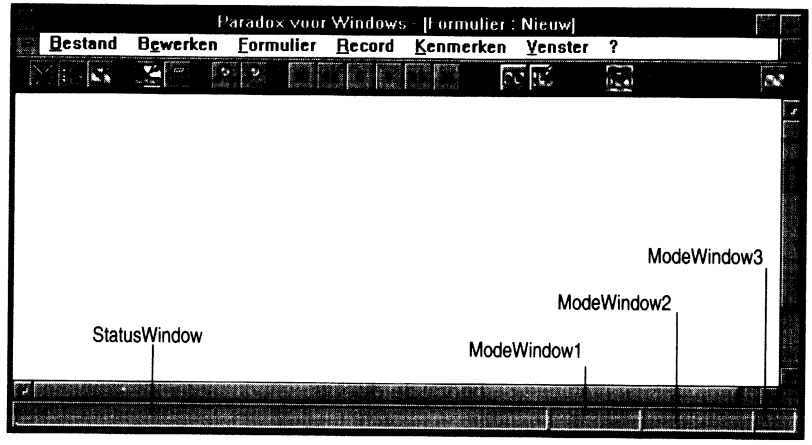
## StatusEvent: de statusbalk besturen

Methodes van het type StatusEvent geven controle over de berichten die worden weergegeven op de statusbalk van het bureaublad. Elk ontwerpobject heeft een ingebouwde **status**-methode. Als u methodes van het StatusEvent-type gebruikt, kunt u code koppelen aan de ingebouwde methode om te bepalen waar berichten worden weergegeven en waarom. U kunt de berichten ook blokkeren of deze in een ander statusgebied of in een ander object weergeven (bijvoor-

beeld in een veldobject of een tekstbestand). U kunt ook bepalen welke tekst in een bericht verschijnt.

U kunt de StatusReasons-constanten ModeWindow1, ModeWindow2, ModeWindow3 en StatusWindow gebruiken om te verwijzen naar de gebieden op de statusregel die u ziet in Afbeelding 12-5. Paradox en ObjectPAL kennen geen beperkingen (behalve de grootte van het gebied) voor de berichten die u in deze gebieden toont. Hoe u de gebieden gebruikt, is uw zaak, maar u kunt het best consequent zijn.

Afbeelding 12-5 De statusbalk




De statusbalk bevat vier vensters. Deze kunt u adresseren met **reason** en **setReason** en de StatusReasons-constanten.

U kunt **reason**, **setReason** en de StatusReasons-constanten gebruiken om te bepalen en op te geven waar een bericht zal worden getoond (zie de bespreking van **reason**, eerder in dit hoofdstuk). Bijvoorbeeld:

```
if eventInfo.reason() = ModeWindow2 then
   eventInfo.setReason(StatusWindow) ; stuur bericht door
endIf
```

Dit voorbeeld kijkt naar berichten die naar het tweede modusgebied gaan en stuurt deze door naar het statusgebied.

***Opmerking*** Als binnen de **status**-methode **message** wordt aangeroepen of een andere code die tekst op de statusbalk weergeeft, wordt er *geen* nieuwe StatusEvent geactiveerd. Zo vermijdt ObjectPAL een eindeloze lus.

U kunt **statusValue** en **setStatusValue** gebruiken om de tekst van het bericht op te vragen en in te stellen. Bijvoorbeeld:

```
method status(var eventInfo StatusEvent)
   if eventInfo.statusValue() = "Eerste record" then
      eventInfo.setStatusValue("Begin") ; verander het bericht
   endIf
endMethod
```

Dit voorbeeld onderschept en verandert het standaardbericht dat verschijnt als u de aanwijzer op de knop 'Eerste record' op de TurboBalk plaatst. Als u de aanwijzer op een knop op de TurboBalk plaatst, wordt standaard de ingebouwde **status**-methode (reason = StatusWindow) geactiveerd en verschijnt een bericht in het statusgebied. Als u de aanwijzer van een knop op de TurboBalk weghaalt, wordt **status** ook geactiveerd (reason = StatusWindow) om het statusgebied leeg te maken.

Het volgende voorbeeld onderschept elk statusbericht en toont het in een dialoogvenster in plaats van op de statusbalk.

```
method status(var eventInfo StatusEvent)
   msgInfo("Bericht:", eventInfo.statusValue())
                       ; toon het bericht in een dialoogvenster
   disableDefault ; voorkom dat de ingebouwde code wordt uitgevoerd
endMethod
```

# TimerEvent: acties na opgegeven intervallen

Methodes van het TimerEvent-type verwerken informatie die wordt gebruikt door de timer-methode die is ingebouwd in ieder ontwerpobject. **isFirstTime** geeft bijvoorbeeld aan of het formulier een bepaalde TimerEvent ziet voordat de actie naar het beoogde doelobject wordt verstuurd, en **getTarget** geeft aan welk object het beoogde doel is.

U gebruikt **setTimer**, gedefinieerd voor het UIObject-type, om te bepalen wanneer timer-acties naar een object moeten worden gezonden, en vervolgens om de ingebouwde **timer**-methode van het object aan te passen om te bepalen hoe het object reageert. (U gebruikt **killTimer**, gedefinieerd voor het UIObject-type, om de timer van een object uit te schakelen.) De volgende voorbeelden van methodes veranderen de positie van een knop elke tiende seconde, zodat het lijkt of de knop over het scherm zweeft.

In deze methodes wordt verondersteld dat u al een formulier hebt gemaakt en een knop hebt getekend. Declareer eerst variabelen in het Var-venster van de knop, zodat alle methodes van de knop de variabelen kunnen zien:

```
var
   posPt Point
   x, y LongInt
endVar
```

Gebruik de volgende code om de **open**-methode van de knop aan te passen.

```
method open(var eventInfo Event)
   self.setTimer(100)         ; genereer elke 100/1000 seconde een TimerEvent
   posPt = self.position      ; position is een kenmerk
```

```
   x = posPt.x()              ; bepaal de x-coördinaat van de positie
   y = posPt.y()              ; bepaal de y-coördinaat van de positie
endMethod
```

Gebruik de volgende code om de **timer**-methode van de knop aan te passen:

```
method timer(var eventInfo TimerEvent)
   x = x + 100
   if x > 5000 then
      x = 10
   endIf
   self.position = Point(x, y) ; ga naar de nieuwe positie
endMethod
```

# ValueEvent: gewijzigde veldwaarden afhandelen

ValueEvent-methodes bepalen wat er gebeurt als de waarde van een veld verandert. De ingebouwde methodes voor veldobjecten kennen twee methodes: **changeValue** en **newValue**. **changeValue** vraagt toestemming om de waarde van een veld te veranderen. Deze methode geeft u de mogelijkheid de waarde te controleren en te beslissen of u deze werkelijk wilt doorvoeren. **newValue** geeft aan wanneer een veld een nieuwe waarde heeft gekregen. Als u bijvoorbeeld een waarde in een veldobject typt en vervolgens de verandering doorvoert (meestal door het veldobject te verlaten), is de volgorde:

```
changeValue
newValue
```

Deze volgorde is anders bij een veld dat wordt weergegeven als keuzeknoppen, als lijst, of als afrollijst voor bewerking. Met deze velden geeft u een waarde op met één handeling—klikken op een keuzeknop of een element uit een lijst kiezen—in plaats van door de reeks toetsaanslagen die nodig is om een waarde te typen. Als u dus met velden van het type keuzeknop, lijst en afrol-bewerken een waarde opgeeft, activeert u **newValue**, **changeValue** en vervolgens weer **newValue**. De eerste **newValue** en de tweede **newValue** hebben echter andere oorzaken.

U kunt **reason** met **eventInfo**- en ValueReason-constanten gebruiken om te testen op de reden voor een **newValue**-methode (maar niet van een **changeValue**-methode), zoals u hieronder ziet. De eerste keer is EditValue de constante, omdat u het veld hebt bewerkt en een nieuwe waarde hebt opgegeven, maar de waarde niet is doorgevoerd. De tweede keer is de constante FieldValue, die aangeeft dat het veld een nieuwe waarde heeft. De volgorde is in pseudocode als volgt:

```
nweWaarde, eventInfo.reason() = EditValue    ; nieuwe waarde opgegeven
; Probeer de verandering door te voeren (bijvoorbeeld door naar een ander
; object te gaan)
changeValue                                  ; vraag toestemming om  waarde  te
```

```
                                                    ; veranderen
newValue, eventInfo.reason() = FieldValue           ; informatie over de nieuwe waarde
```

Als een formulier wordt geopend, wordt bovendien voor elk object op het formulier **newValue** geactiveerd; de reason-constante is StartupValue.

***Belangrijk*** Een berekend veld toont alleen waarden, maar schrijft deze niet naar een tabel. Paradox schrijft de gegevens uit een berekend veld nooit naar een tabel, zodat de **changeValue**-methode van het veld nooit wordt aangeroepen. **newValue** wordt echter wel elke keer aangeroepen als het veld een nieuwe waarde toont. Zie Appendix B voor meer informatie. Stel dat in het volgende voorbeeld een formulier een veldobject insluit dat is verbonden met het veld 'Ordernr..' van de tabel *Order* en dat het formulier een knop insluit met de naam *sendError*. De **pushButton**-methode voor *sendError* maakt een ValueEvent en stelt de foutcode voor die actie in op CanNotDepart. De actie wordt dan naar de **changeValue**-methode voor *Ordernr.* gestuurd.

***Opmerking*** De naam van een veld in een tabel kan punten bevatten, maar de naam van een object in een formulier niet. Paradox vervangt een punt standaard door een onderstrepingsteken. In dit voorbeeld verwijst 'Ordernr.' dus naar het veld in de tabel en *Ordernr_* naar het veldobject in het formulier.

```
; sendError::pushButton
method pushButton(var eventInfo Event)
var
  va  ValueEvent              ; te versturen event
  ui  UIObject
endVar
va.setErrorCode(CanNotDepart) ; stel fout in
ui.attach(Ordernr_)
ui.changeValue(va)            ; verzend event
endmethod
```

Deze code is gekoppeld aan de **changeValue**-methode voor het veldobject *Ordernr_*. In dit voorbeeld rapporteert de methode alleen de fout.

```
; Ordernr_::changeValue
method changeValue(var eventInfo ValueEvent)
  msgInfo("En de fout was...",
          String(eventInfo.errorCode()))
endmethod
```

Stel dat u in het volgende voorbeeld wilt weten of de gebruiker een veld in een record heeft veranderd. Een manier om dit te doen is de waarde van het kenmerk 'Touched' van het record te controleren. 'Touched' wordt echter True als de gebruiker een veld bewerkt, maar het geen *nieuwe* waarde geeft.

Als u wilt weten of de oude waarde verschilt van de nieuwe, kunt u deze waarden vergelijken in de **changeValue**-methode op het formulier en alleen een vlag instellen als de waarden verschillend zijn. In het volgende voorbeeld wordt verondersteld dat een formulier een multi-record object heeft dat is verbonden met de tabel ORDER.DB, en een knop met de naam *ongedaanKnop*. Als de variabele *OngedaanVlag* True is, worden veranderingen in het record geannuleerd als er op de knop *Ongedaan* wordt gedrukt.

```
; ditFormulier::Var
Var
  OngedaanVlag Logical
endVar
```

Deze methode wordt gekoppeld aan de **open**-methode van het formulier:

```
; ditFormulier::open
method open(var eventInfo Event)
if eventInfo.isPreFilter()
  then
    ;deze code wordt uitgevoerd voor alle objecten op het formulier
  else
    ;deze code wordt alleen voor het formulier uitgevoerd
    UndoFlag = False
endif
endmethod
```

Deze methode wordt gekoppeld aan de **changeValue**-methode van het formulier:

```
; ditFormulier::changeValue
method changeValue(var eventInfo ValueEvent)
var
  doelObj  UIObject              ; bepaal het doel van de actie
endVar
if eventInfo.isPreFilter()
  then
    ;deze code wordt uitgevoerd voor alle objecten op het formulier
    eventInfo.getTarget(doelObj)       ; bepaal het doel van de actie
    if doelObj.Value <> eventInfo.newValue() then
       OngedaanVlag = True
       OngedaanVlag.view()
    endif
  else
    ;deze code wordt alleen voor het formulier uitgevoerd

endif
endmethod
```

Deze code wordt gekoppeld aan de **pushButton**-methode van *ongedaanKnop*:

```
; ongedaanKnop::pushButton
method pushButton(var eventInfo Event)
if OngedaanVlag then                    ; controleer of record is veranderd
  ORDER.action(DataCancelRecord)
  OngedaanVlag = False
else
  msgInfo("Status", "Geen veranderingen om te annuleren.")
endif
; u kunt ook het kenmerk 'Touched' van het MRO bekijken
endmethod
```

HOOFDSTUK 13

# Ontwerpobjecten

Ontwerpobjecten vormen de gebruikersinterface van een applicatie. Alles wat u in een formulier kunt plaatsen, is een ontwerpobject. Er zijn drie soorten ontwerpobjecten: menu's, pop-up menu's en UIObjecten. UIObjecten omvatten objecten die met de ontwerp-hulpmiddelen van de TurboBalk zijn gemaakt, recordobjecten, de onderliggende pagina van een formulier en meer. UIObjecten zijn de enige objecten met ingebouwde methodes, de enige objecten waaraan u code kunt koppelen en de enige objecten die op acties reageren. In Tabel 13-1 wordt een beschrijving gegeven van alle ontwerpobjecten.

***Opmerking*** Deze categorie omvat ook het formulier, hoewel het type Form zelf een weergavebeheer-object is en geen ontwerpobject. Een formulier heeft ingebouwde methodes waaraan u code kunt koppelen. Een formulier reageert dus op acties.

Tabel 13-1 Ontwerpobjecten

| Type | Beschrijving |
|---|---|
| UIObject | Objecten die op een formulier zijn geplaatst om de gebruikersinterface te vormen. De lidobjecten zijn bitmap, kader, knop, kruistabulatie, ellips, veldobject, formulier, grafiek, lijn, multi-record object, OLE-object, pagina, recordobject, tabelframe, en tekstvak. |
| Menu | Menu's die op de menubalk van de applicatie verschijnen. |
| PopUpMenu | Menu's die op een formulier verschijnen, niet op de menubalk van de applicatie. |

In dit hoofdstuk komen de volgende onderwerpen aan de orde:

- UIObjecten: bouwstenen van de gebruikersinterface
- Menu: een lijst boven in het venster
- PopUpMenu: lijsten op verzoek
- Werken met ingebouwde menu's en de TurboBalk

# UIObjecten: bouwstenen van de gebruikersinterface

Het UIObject-type (UI staat voor user interface: gebruikersinterface) omvat alle objecten die u kunt plaatsen met de hulpmiddelen van de TurboBalk, (beschreven in het *Handboek*), de onderliggende pagina, het formulier en meer. Een volledige lijst met UIObjecten is online beschikbaar. Als u de lijst wilt weergeven, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Kenmerken'. De UIObjecten worden in de kolom 'Objecten' weergegeven.

In deze paragraaf worden de volgende zaken besproken:

- Kenmerken van UIObjecten
- Het gebruik van het kenmerk 'Value' om gegevens op te vragen en in te stellen
- Het gebruik van kenmerken die zijn verbonden met ingebouwde methodes
- Kenmerken: speciale gevallen
- Samengestelde objecten
- Acties en UIObjecten
- Handelingen en UIObjecten
- Handelingen en tabelframes
- Handelingen en records
- UIObjecten: snelle manieren en speciale gevallen
- Werken met ObjectPAL in berekende velden

## Kenmerken

Alle objecten hebben, naast methodes, *kenmerken* (bijvoorbeeld tekst, kleur, patroon, font, naam, grootte en positie). Kenmerken lijken op variabelen, maar worden voorgedefinieerd als onderdelen van objecten. Dat wil zeggen, bij elk object horen ingebouwde kenmerken. Als u een object inspecteert, verschijnt er een lijst met de bijbehorende kenmerken. U kunt met ObjectPAL toegang krijgen tot deze kenmerken. Elk kenmerk dat u vanuit het menu kunt instellen, kunt u ook met behulp van ObjectPAL instellen. Bovendien kunt u met ObjectPAL veel kenmerken instellen die niet in menu's verschijnen.

***Opmerking*** Als u een kenmerk instelt, wordt er geen actie gegenereerd, *behalve* als u het kenmerk 'Value' van een veldobject instelt (hierdoor wordt de **newValue**-methode geactiveerd).

De meeste kenmerken hebben invloed op het uiterlijk van een object. De volgende instructie stelt bijvoorbeeld het kenmerk 'Text' van het

tekstvak *mijnTekst* zo in dat het woord "Annuleren" wordt weergegeven:

```
mijnTekst.text = "Annuleren"
```

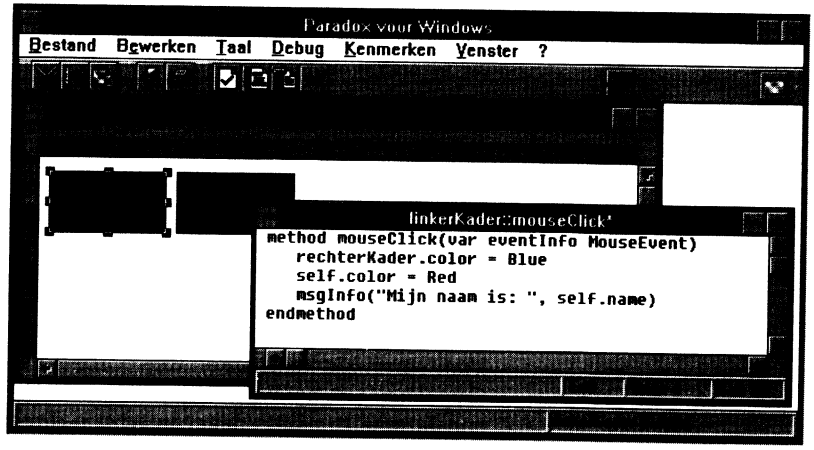
Stel dat een formulier een veldobject met de naam *resultaat* insluit. U kunt het kenmerk 'Color' gebruiken om voor de tekens een kleur op te geven die afhankelijk is van bepaalde gegevens, bijvoorbeeld:

```
if inkomen > uitgaven then
   resultaat.font.color = Green ; gebruik groene tekens in geval van winst
else
   resultaat.font.color = Red ; als er geen winst is, gebruik dan rode tekens
endif
```

Objecten kunnen eigen kenmerken en de kenmerken van andere objecten instellen. In Afbeelding 13-1 wordt bijvoorbeeld een formulier getoond dat twee kaders bevat. Aan het linkerkader (*linkerKader*) is een methode gekoppeld die het eigen kenmerk 'Color' en het kenmerk 'Color' van het rechterkader (*rechterKader*) verandert. Vervolgens geeft de methode de naam van het kader ('Name' is een kenmerk) in een dialoogvenster weer.

```
; deze methode is gekoppeld aan linkerKader
method mouseEnter (var eventInfo MouseEvent)
   rechterKader.color = Blue       ; stelt de kleur van rechterKader in
   self.color = Red                ; stelt de kleur van dit object (linkerKader)
                                   ; in
   msgInfo("Ik heet", self.name)   ; geeft de naam van dit object (linkerKader)
                                   ; weer
endMethod
```

Afbeelding 13-1 Een formulier met twee kaders




De methode in het Editor-venster wordt uitgevoerd als u de aanwijzer in het linkerkader plaatst. De methode stelt de kleur van het rechterkader in op blauw en van het linkerkader op rood en geeft een dialoogvenster weer dat de naam van het linkerkader bevat.

Zie Appendix C voor meer informatie over de objectkenmerken. Objectkenmerken en kenmerkwaarden zijn ook online beschikbaar.

## Kenmerken gebruiken

Via ObjectPAL hebt u toegang tot veel meer objectkenmerken dan wanneer u Paradox interactief gebruikt. U gebruikt puntnotatie om direct toegang te krijgen tot een kenmerk, zoals in het volgende voorbeeld:

```
if mijnPagina.color = Red then ; directe toegang tot kleurkenmerk
   mijnPagina.color = Green
   mijnKader.visible = No ; maakt mijnKader onzichtbaar
endIf

; de volgende twee regels geven Strings terug
kenmVisible = mijnKader.visible
kenmColor = mijnKader.color
mijnVeld.font = "System"  ; stelt het font van mijnVeld in op System
mijnVeld.font.color = Red ; stelt de font-kleur in op Rood
datKader.visible = not(datKader.visible) ; schakelt tussen zichtbaar en
                                         ; onzichtbaar
```

## Gegevenstypes van kenmerken

Elk kenmerk heeft een natuurlijk gegevenstype. Het natuurlijke gegevenstype van het kenmerk 'Color' is LongInt (lang geheel getal). U hoeft echter geen waarden van gehele getallen te onthouden om kleuren in te stellen, omdat ObjectPAL voorgedefinieerde constanten kent die veel (maar niet alle) kenmerkwaarden vertegenwoordigen. De volgende instructie werkt omdat ObjectPAL het woord 'Blue' heeft gedefinieerd als een constante waarvan de waarde 16.711.680 is (de geheel-getalwaarde van de kleur blauw).

```
datKader.color = Blue
datKader.Frame.Style = Shadow
```

**Opmerking** U kunt kenmerken ook toewijzen met een kenmerkconstante in een reeks tussen aanhalingstekens of door de waarde van een constante aan een variabele toe te wijzen. Alle volgende instructies zijn bijvoorbeeld geldig:

```
var hetKenmerk AnyType endVar

datKader.Frame.Style = Shadow          ; gebruikt een constante
datKader.Frame.Style = "Shadow"        ; gebruikt een constante als een reeks
                                       ; tussen aanhalingstekens

hetKenmerk = Shadow
datKader.Frame.Style = hetKenmerk      ; gebruikt een variabele die aan een
                                       ; constante is toegewezen
```

Deze werkwijzen kunnen handig zijn als u kenmerkwaarden vanuit een externe bron krijgt, bijvoorbeeld vanuit een tabel of een tekstbestand.

Voor sommige kenmerken (bijvoorbeeld voor 'Text', 'LabelText' en 'TypeFace'), is het natuurlijke type String. Er zijn dus geen constanten. In plaats daarvan wijst u direct een waarde toe, bijvoorbeeld als volgt:

```
tekstVak.Text = "Voer het onderdeelnummer in."
mijnKnop.LabelText = "Annuleren"
mijnVeldObject.Font.TypeFace = "Times"
```

Zie Appendix C voor een volledige lijst met kenmerken en de bijbehorende gegevenstypes.

***Opmerking*** De maximale lengte van een reeks die als een kenmerkwaarde wordt teruggegeven, is 255 tekens, behalve voor het kenmerk 'Text', dat 32.767 tekens kan bevatten.

### *Self*

U kunt de objectvariabele *Self* gebruiken in een methode: *Self* verwijst naar het object waaraan de code die wordt uitgevoerd, is gekoppeld. De volgende instructie wordt uitgevoerd in de methode **mouseEnter**, die is gekoppeld aan *linkerKader* en *Self* verwijst dus naar *linkerKader*:

```
self.color = Red
```

Stel echter dat een methode die aan *linkerKader* is gekoppeld, de eigen methode **wijzigKleur** aanroept, die is gekoppeld aan de pagina, en dat de code voor **wijzigKleur** als volgt luidt:

```
method wijzigKleur()
   self.color = Blue
endMethod
```

Als de methode die aan *linkerKader* is gekoppeld, **wijzigKleur** aanroept, wordt de pagina blauw, maar *linkerKader* niet. Waarom is dat? Omdat *Self* verwijst naar het object waaraan de code is gekoppeld, ongeacht welk object de code eigenlijk heeft aangeroepen en in dit geval is dat de code die aan de pagina is gekoppeld.

Zie Appendix B voor meer informatie over *Self* en andere ingebouwde variabelen.

***Opmerking*** De enige uitzondering op deze regel is de situatie waarin een methode in een bibliotheek wordt aangeroepen. De code in een bibliotheek wordt uitgevoerd namens het object dat de code heeft aangeroepen. Als een kader bijvoorbeeld een bibliotheekmethode aanroept, verwijzen instructies die *Self* gebruiken, naar het kader en niet naar de bibliotheek. Zie de paragraaf "Library" in Hoofdstuk 17 voor meer informatie.

*Self* is bovendien niet hetzelfde als *Subject*. Subject is een objectvariabele die verwijst naar de aanroeper van een eigen methode. Zie Hoofdstuk 11 voor meer informatie.

## Met het kenmerk 'Value' kunt u waarden opvragen en instellen

Een van de belangrijkste objectkenmerken is *Value*. Met het kenmerk 'Value' kunt u gegevens in een veld of een tabel opvragen of instellen, zoals in dit voorbeeld:

```
x = tabelID.veldID.value      ; vraagt de veldwaarde op
tabelID.veldID.value = z      ; stelt de veldwaarde in
```

Het gedeelte *.value* is optioneel. U kunt *.value* toevoegen voor de duidelijkheid of weglaten voor de snelheid. Met andere woorden, de volgende instructies zijn in ObjectPAL identiek:

```
x = tabelID.veldID

x = tabelID.veldID.value
```

Stel dat u de voornaam van de klant Jansen wilt weten. U kunt als volgt in het veld 'Achternaam' van het tabelframe *KLANT* naar Jansen zoeken en de waarde van het veld 'Voornaam' opvragen:

```
var
   voorNaam String
endVar
if KLANT.locate("Achternaam", "Jansen") then ; als Jansen in de tabel staat,
                                             ; plaats dan
   voorNaam = KLANT.Voornaam.value           ; waarde van
                                             ; het veld 'Voornaam'
endIf                                        ; in de variabele voorNaam
msgInfo("Voornaam", voorNaam)                ; toon de naam in een
                                             ; dialoogvenster
```

Bij tekstobjecten, inclusief veldlabels en knoplabels, stelt het kenmerk 'Value' de tekst in het object in. U kunt hetzelfde effect bereiken door het kenmerk 'Text' in te stellen. De volgende instructies hebben bijvoorbeeld hetzelfde effect:

```
mijnTekst.Value = "Hallo"

mijnTekst.Text = "Hallo" ; deze instructies zijn identiek
```

Knoppen hebben ook een 'Value'-kenmerk. Als de waarde van een knop True is, lijkt de knop te worden ingedrukt (of is de knop geselecteerd, als het gaat om een aankruisvak of een keuzeknop). Als de waarde False is, springt de knop terug (of wordt deze gedeselecteerd). Deze instructies zorgen ervoor dat de knop *dieKnop* wordt ingedrukt en terugspringt, maar activeren de **pushButton**-methode *niet*:

```
dieKnop.value = True                    ; laat knop indrukken
sleep(1000)
dieKnop.value = False                   ; laat knop terugspringen
```

***Opmerking*** Als u het kenmerk 'Value' van een veldobject instelt, activeert u de **newValue**-methode. De **changeValue**-methode wordt geactiveerd als de nieuwe waarde wordt doorgevoerd.

## Kenmerken die zijn verbonden met ingebouwde methodes

In deze paragraaf wordt een aantal methodes en kenmerken beschreven die zijn verbonden met ingebouwde methodes. In het eerste gedeelte worden kenmerken besproken die alle UIObjecten gemeen hebben, in het tweede gedeelte worden de kenmerken van veldobjecten besproken en in het derde gedeelte de kenmerken van recordobjecten.

**Opmerking** De kenmerken die hier worden besproken, vertegenwoordigen een klein deel van de kenmerken die ObjectPAL kent. Appendix C bevat een volledige lijst met kenmerken voor alle objecten.

### Alle UIObjecten

- *Arrived* is True als de **arrive**-methode van het actieve object wordt aangeroepen. Als 'Arrived' voor een object True is, is het ook True voor de objecten die het object insluiten. Stel dat *veldEen* zich in *paginaEen* bevindt en *veldTwee* in *paginaTwee*. Als 'Arrived' True is voor *veldEen*, is dit kenmerk ook True voor *paginaEen*, maar niet voor *paginaTwee*. Als 'Arrived' True is voor *veldTwee*, is het ook True voor *paginaTwee*, maar niet voor *paginaEen*. 'Arrived' is een alleen-lezen kenmerk (u kunt de waarde opvragen, maar niet instellen).
- *Focus* wordt ingesteld op True als de **setFocus**-methode van het actieve object wordt aangeroepen en op False als de **removeFocus**-methode van het actieve object wordt aangeroepen. Net als bij 'Arrived' geldt dat als 'Focus' True is voor een object, het kenmerk ook True is voor de objecten die het object insluiten. 'Focus' is een alleen-lezen kenmerk.

### Veldobjecten

- *Editing* wordt op True ingesteld als een bewerkvenster van een veld open is. Als u een veldobject bewerkt, maakt Paradox een tekstobject over het veldobject heen. Dit tekstobject is de plaats waar u feitelijk waarden typt. Paradox verwijdert het tekstobject als u klaar bent met de bewerking. Dit kenmerk wordt ingesteld door de ingebouwde **arrive**-methode van het veldobject. 'Editing' is een alleen-lezen kenmerk.
- *Touched* wordt op True ingesteld als de gebruiker de gegevens in een veld wijzigt. Dit kenmerk bepaalt of de inhoud van het bewerkvenster naar het veld wordt geschreven. Als 'Touched' False is, hoeven dezelfde gegevens niet naar het veld te worden geschreven. 'Touched' is een alleen-lezen kenmerk en u kunt het alleen lezen als 'Editing' True is.

### Recordobjecten

- *BlankRecord* geeft True terug als een record leeg is; anders wordt False teruggegeven.
- *Inserting* geeft True terug als een record zojuist in de tabel is ingevoegd; anders wordt False teruggegeven.
- *Locked* geeft True terug als een record is vergrendeld; anders wordt False teruggegeven.
- *RowNo* is een geheel getal dat het rijnummer (vanaf 1) vertegenwoordigt van een record dat in een tabelframe of multi-record object wordt weergegeven. De tweede rij die in een tabelframe wordt weergegeven, heeft bijvoorbeeld een *RowNo* van 2,

ongeacht welk record van de tabel in die rij wordt weergegeven. Zie ook het kenmerk *RecNo* voor tabelframe-objecten en multi-record objecten.

***Opmerking***

Als u *RowNo* test voor een multi-record object, moet u onthouden dat records van boven naar beneden of van links naar rechts kunnen zijn genummerd.

- *Touched* is True als het record is gewijzigd, maar niet doorgevoerd. 'Touched' is een alleen-lezen kenmerk.

***Tabelframes en multi-record objecten***

- *RowNo* is een geheel getal dat het rijnummer van het actieve record vertegenwoordigt. Met dit kenmerk kunt u gemakkelijk bepalen welk relatieve record actief is. 'RowNo' geeft het rijnummer terug ten opzichte van het tabelframe en niet ten opzichte van de onderliggende tabel.
- *SeqNo* is een geheel getal dat het recordnummer van de onderliggende tabel vertegenwoordigt.

***Lijsten met kenmerken***

Als u de lijst wilt zien, opent u een venster van de ObjectPAL-Editor. Vervolgens kiest u 'Taal | Kenmerken' en selecteert u het object en het gewenste kenmerk.

Als u bijvoorbeeld de lijst met kaderkenmerken wilt bekijken, opent u een venster van de ObjectPAL-Editor. Vervolgens kiest u 'Taal | Kenmerken' en 'Box' in de kolom 'Objecten'. De kaderkenmerken worden dan in de kolom 'Kenmerken' weergegeven. Als u de geldige waarden voor een kenmerk wilt zien, kiest u de kenmerknaam (bijvoorbeeld 'Color'). De waarden worden in de kolom 'Waarden' weergegeven.

In Appendix C wordt een overzicht gegeven van de UIObject-kenmerken. De kenmerken zijn ook online beschikbaar.

## Kenmerken: speciale gevallen

In deze paragraaf worden enkele alternatieve manieren getoond voor het manipuleren en instellen van kenmerken van bepaalde ontwerpobjecten. De volgende onderwerpen komen aan de orde:

- Methodes (in plaats van puntnotatie) gebruiken om kenmerken in te stellen
- Het kenmerk 'LabelText' van een object gebruiken om het tekstlabel te veranderen terwijl een applicatie actief is
- Kenmerken van veldobjecten

***Kenmerken instellen met methodes***

U kunt de methodes **getProperty**, **getPropertyAsString** en **setProperty** van het UIObject-type gebruiken als alternatief voor de puntnotatie. Deze methodes zijn handig omdat deze de kwestie van

de gegevenstypes omzeilen en alle kenmerkwaarden als reeksen behandelen. Stel dat een formulier een kader bevat dat *datKader* heet. De volgende code manipuleert het kenmerk 'Frame.Style' van het kader:

```
var
   kenmerkNaam, kenmerkWaarde String
endVar
kenmerkNaam = "frame.style"
kenmerkWaarde = datKader.getPropertyAsString(kenmerkNaam)
if kenmerkWaarde <> "ShadowFrame" then
   datKader.setProperty(kenmerkNaam, ShadowFrame)
endIf
```

### Kenmerken van knopobjecten

Knoppen hebben een kenmerk, 'LabelText', waarmee u de tekst in het label kunt opgeven zonder het label te benoemen (het is een tekstobject) en het direct te adresseren. De volgende methode selecteert bijvoorbeeld het kenmerk 'LabelText' en stelt het voor alle drie de knoppen in:

```
var
   knoppen Array[3] String
   i SmallInt
endVar
knoppen[1] = "knopEen"
knoppen[2] = "knopTwee"
knoppen[3] = "knopDrie"
for i from 1 to knoppen.size()
   if knoppen[i].labelText = "Aan" then
      knoppen[i].labelText = "Uit"
   else
      knoppen[i].labelText = "Aan"
   endIf
endFor
```

### Kenmerken van veldobjecten

Veldobjecten hebben veel kenmerken. Hieronder wordt slechts een klein aantal genoemd.

- *CursorLine* rapporteert de verticale positie van de invoegpositie in een veldobject dat met de eerste regel is verbonden. De eerste regel is 1, de tweede 2 enzovoort.
- *CursorCol* geeft de kolom van de invoegpositie terug, dat wil zeggen, het aantal tekens tussen de invoegpositie en de linkermarge van het veld.
- *CursorPos* geeft het aantal tekens terug tussen de invoegpositie en het eerste teken van het veld.
- *SelectedText* geeft een reeks terug die de huidige geselecteerde tekst vertegenwoordigt of een lege reeks als er geen tekst is geselecteerd.

  Stel dat een veld de volgende tekst bevat:

  ```
  "Selecteer het derde woord van deze zin"
  ```

Stel vervolgens dat de **mouseRightUp**-methode van dit veld als volgt luidt:

```
method mouseRightUp(var eventInfo MouseEvent)
if self.selectedText <> "" then
   msgInfo("U selecteerde", self.selectedText)
endIf
endMethod
```

Als u dit formulier start, het derde woord van de zin selecteert en rechts klikt, verschijnt er een dialoogvenster. De titel is "U selecteerde" en in het dialoogvenster bevindt zich het woord "derde".

- *FieldName* rapporteert met welk veld van welke tabel een veldobject is verbonden. De volgende instructie verbindt bijvoorbeeld het veldobject *mijnVeld* met het veld 'Achternaam' van de tabel *Klant* (die moet zijn opgenomen in het gegevensmodel van het formulier).

  ```
  mijnVeld.fieldName = "[Klant.Achternaam]"
  ```

  De aanhalingstekens om de vierkante haken zijn vereist. Als u naar een tabel verwijst met de bestandsextensie (bijvoorbeeld KLANT.DBF) plaatst u aanhalingstekens, voorafgegaan door backslash-tekens, om de naam van de tabel, zoals in de volgende instructie:

  ```
  mijnVeld.fieldName = "[\"klant.dbf\".Achternaam]"
  ```

  Dit is noodzakelijk omdat de punt in de bestandsextensie een speciaal teken is voor de ObjectPAL-compiler en uw code anders verkeerd wordt geïnterpreteerd. In het algemeen gebruikt u aanhalingstekens, voorafgegaan door backslash-tekens, waar punten voor onduidelijkheid kunnen zorgen.

  Voor spaties in een veldnaam zijn geen aanhalingstekens vereist, bijvoorbeeld:

  ```
  mijnVeld.fieldName = "[Klant.Eerste contact]"
  ```

  Ook het kenmerk 'DataSource', dat een veld als een bron voor elementen in een lijst opgeeft, gebruikt deze syntaxis.

- *Value* vraagt de huidige waarde van een veldobject op of stelt deze in.

  Veldobjecten hebben een kenmerk met de naam 'Value' dat de waarde krijgt van AnyType. De volgende instructies wijzen bijvoorbeeld aan *x* de waarde van het veldobject *eenVeld* toe, ongeacht het type gegevens dat *eenVeld* bevat:

  ```
  var x AnyType endVar

  x = eenVeld.value
  ```

U kunt het sleutelwoord VALUE voor het gemak weglaten in uitdrukkingen. De volgende instructies zijn dus identiek:

```
x = eenVeld.value

x = eenVeld
```

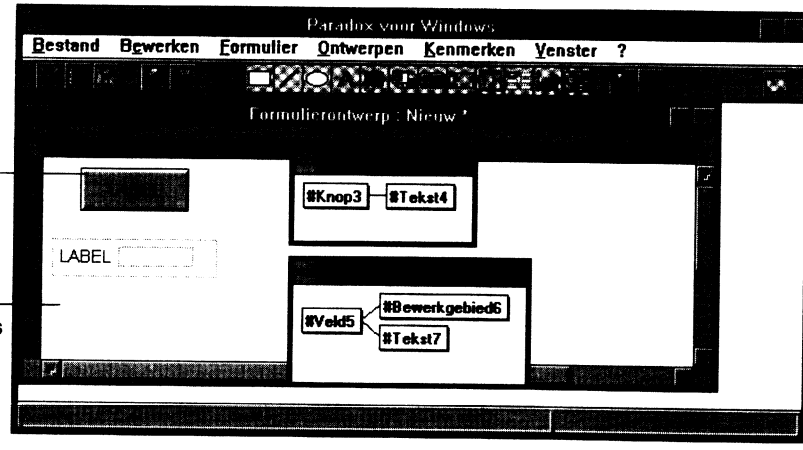
Onthoud dat u, zelfs als u het sleutelwoord VALUE niet gebruikt, met de waarde van het object werkt en niet met het object zelf.

## Samengestelde objecten

Een samengesteld object bestaat uit twee of meer objecten. Een eenvoudig voorbeeld is een knop, die bestaat uit het knopobject en een tekstvak dat als een label fungeert. Een ander voorbeeld is een veld met een label, dat bestaat uit het veldobject, het label en het bewerkgebied. U kunt het Objectenschema gebruiken om te zien hoe objecten in een samengesteld object met elkaar zijn verbonden. Selecteer het object en kies vervolgens 'Formulier | Objectenschema'. In Afbeelding 13-2 ziet u de schema's voor een knop en een veld met een label.

***Opmerking*** Als u met velden met een label werkt, koppelt u methodes normaal gesproken aan het veldobject en niet aan het label of aan het bewerkgebied.

Afbeelding 13-2 Objectenschema van een knop en een veld met een label



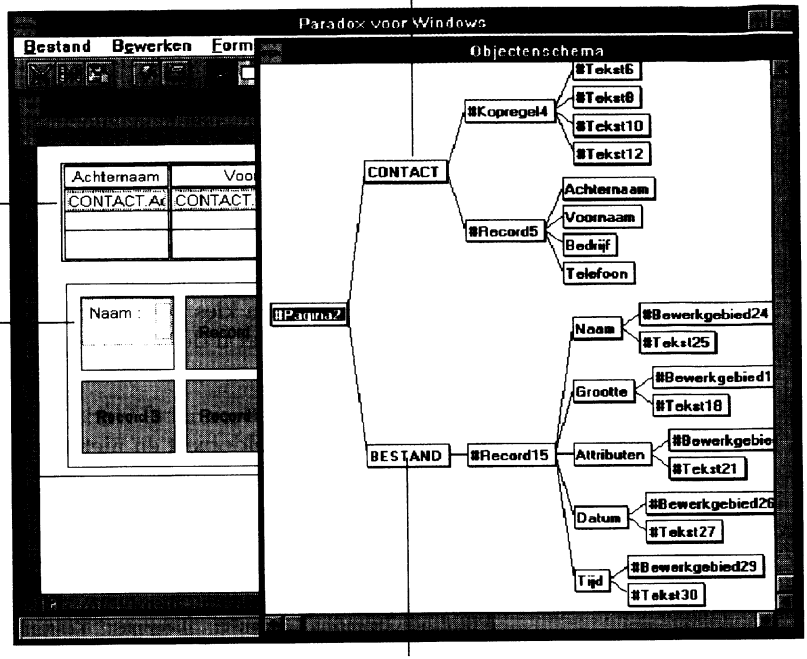

Het schema van deze knop toont het knopobject en het tekstvak dat als label dient. U koppelt methodes normaal gesproken aan het knopobject (in het schema gemarkeerd).

Het schema van dit veld met een label toont het veldobject, het tekstvak dat als label dient, en het bewerkgebied waarin de gegevens worden geplaatst. U koppelt methodes aan het veldobject (in het schema gemarkeerd).

Tabelframes en multi-record objecten zijn complexere samengestelde objecten. Zoals u in Afbeelding 13-3 ziet, sluit een tabelframe een recordobject in dat veldobjecten kan insluiten die weer tekstvakken kunnen insluiten. Al deze objecten worden automatisch gemaakt, als u het tabelframe maakt.

Afbeelding 13-3 Objectenschema voor een tabelframe en een multi-record object

Dit deel van het schema toont de objecten in het tabelframe CONTACT. Let op het recordobject dat de veldobjecten insluit. Alles wat u met het recordobject doet, is van invloed op alle records in het tabelframe; alles wat u met een veldobject doet, is van invloed op dat veld in alle records.




Dit is een tabelframe dat is verbonden met CONTACT.DB.

Dit is een multi-record object dat is verbonden met BESTAND.DB. Er worden vier records uit de tabel weergegeven. De grijze vakken zijn slechts plaatshouders die u niet kunt inspecteren.

Dit deel van het schema toont de objecten in het multi-record object BESTAND. Dit multi-record object geeft vier records weer die alle dezelfde structuur hebben. Het schema toont alleen de objecten die zich in één record bevinden. Alles wat u met het record doet, is van invloed op alle records en alles wat u doet met een object in het record, is van invloed op dat object in alle records. De andere records in het schema zijn slechts plaatshouders, de tegenhangers van de grijze vakken in het multi-record object.

Hoewel er geen hulpmiddel op de TurboBalk bestaat om record-objecten (in tegenstelling tot multi-record objecten) te maken, behoren recordobjecten tot het UIObject-type. U kunt met alle objecten (inclusief recordobjecten) in een samengesteld object werken, zoals u dat ook met alle andere objecten van het UIObject-type kunt doen. U kunt het object inspecteren, de kenmerken instellen en methodes koppelen. De afbeelding toont ook hoe multi-record objecten plaatshouders gebruiken die meerdere records vertegenwoordigen.

Een object dat met een tabel is verbonden, ontleent de naam standaard aan die tabel. Als u bijvoorbeeld een tabelframe met de tabel *Klant* verbindt, wordt de naam van het tabelframe *KLANT*. U kunt de standaardnaam wijzigen door het object interactief te inspecteren en een nieuwe naam in te voeren of door ObjectPAL te

gebruiken om het kenmerk 'Name' te veranderen. In beide gevallen kunt u het kenmerk 'TableName' gebruiken om de naam van de onderliggende tabel te bepalen. Zie het *Handboek* voor meer informatie over tabelframes en multi-record objecten (evenals kruistabulaties en grafieken).

## Acties en UIObjecten

UIObjecten reageren op alle soorten acties. Verschillende objecten kunnen anders reageren op dezelfde actie. Zie Appendix B voor meer informatie.

### Toetsenbordacties

U kunt de ingebouwde methodes **keyChar, keyPhysical** en **action** gebruiken om op toetsenbordacties te reageren. Zie Hoofdstuk 12 voor meer informatie en voorbeelden.

U gebruikt de methode **keyChar**, die is gedefinieerd voor het UIObject-type, om tekens naar een object in een formulier te sturen. De volgende code stuurt bijvoorbeeld de letter *A* naar het veldobject *hester*. U kunt deze code aan een knop koppelen. Elke keer als u dan op de knop drukt, verschijnt er een *A* in het veldobject.

```
method pushButton(var eventInfo Event)
   hester.keyChar("A")
endMethod
```

U kunt **keyChar** ook gebruiken om tekenreeksen te versturen. De volgende code stuurt bijvoorbeeld de reeks "To be, or not to be" naar het veldobject *hamlet*.

```
method pushButton(var eventInfo Event)
   hamlet.keyChar("To be, or not to be")
endMethod
```

### Muisacties

UIObjecten hebben ingebouwde methodes die u kunt wijzigen, zodat deze methodes reageren op veel voorkomende klikken en verplaatsingen van de muis. U kunt ook **hasMouse** gebruiken om te bepalen of de aanwijzer zich op een object bevindt en **wasLastClicked** en **wasLastRightClicked** gebruiken om te bepalen of een object het laatste object was waarop met de muis is geklikt (of waarop rechts is geklikt). Zie Hoofdstuk B en Appendix 12 voor meer informatie en voorbeelden.

### Timer-acties

U gebruikt **setTimer** om aan te geven wanneer timer-acties naar een object moeten worden gestuurd. Vervolgens wijzigt u de **timer**-methode van het object om te bepalen hoe het object reageert. (Gebruik **killTimer** om de timer van een object uit te schakelen.) In het volgende voorbeeld worden bitmaps weergegeven na opgegeven timer-intervals, om een animatie-effect te bewerkstelligen.

Deze methodes gaan ervanuit dat u al een formulier hebt gemaakt en twee bitmap-objecten hebt geplaatst (namelijk *klapOmhoog* en *klapOmlaag*), waarbij het ene object op het andere is geplaatst.

Gebruik de volgende code om de **open**-methode van het formulier te wijzigen:

```
method open(var eventInfo Event)
   self.setTimer(100)            ; genereer elke 100/1000 van een seconde een
TimerEvent
   klapOmhoog.visible = True     ; geef één bitmap weer
   klapOmlaag.visible = False    ; verberg de andere bitmap
endMethod
```

Gebruik de volgende code om de **timer**-methode van het formulier te wijzigen:

```
method timer(var eventInfo TimerEvent)
   klapOmhoog.visible = not(klapOmhoog.visible) ; wissel het kenmerk 'Visible'
   klapOmlaag.visible = not(klapOmlaag.visible)
endMethod
```

### Waarde-acties

Waarde-acties vinden plaats als de waarde van een veld verandert. ObjectPAL kan onderscheid maken tussen het schuiven van het ene record naar het andere en het invoeren van gegevens in een veld. U kunt ook de **newValue**-methode van het ValueEvent-type in de ingebouwde **changeValue**-methode van een veldobject gebruiken om de waarde van een veldobject te inspecteren en te bepalen of u de waarde wilt doorvoeren.

Stel dat de volgende code aan de ingebouwde **changeValue**-methode van een veldobject is gekoppeld. Als u in dit veld een waarde invoert en de waarde probeert door te voeren (bijvoorbeeld door op *Tab* of op *Return* te drukken), inspecteert deze methode de waarde. Als de waarde kleiner is dan 100, opent deze methode een dialoogvenster waarin u wordt gevraagd een andere waarde in te voeren. Anders verwerkt de ingebouwde standaardcode de waarde op de gebruikelijke manier.

```
method changeValue(var eventInfo ValueEvent)
   if eventInfo.newValue() < 100 then
      msgStop("Stop", "Voer een waarde in die groter is dan 100.")
      disableDefault
   endIf
endMethod
```

Raadpleeg de paragraaf "ValueEvent" in Hoofdstuk 12 en de online ObjectPAL Help voor meer informatie en voorbeelden.

## Demonstratie: acties, objecten en insluiting

*Stap 1: aan de slag*

In deze paragraaf worden de stappen getoond die nodig zijn om een formulier te maken en methodes te schrijven die de relatie tonen tussen objecten, insluiting en acties.

1. Kies 'Bestand | Nieuw | Formulier' en accepteer de standaard-instellingen om een leeg formulier te maken.
2. Teken met het Kader-hulpmiddel in de linkerbovenhoek van het formulier een kader waarvan de zijden ongeveer vier centimeter lang zijn.
3. Inspecteer het kader en verander de naam in *linkerBuitenKader*.
4. Selecteer het kader en druk vervolgens op *Ctrl-Spatiebalk* om het methodevenster te openen.
5. Kies **canArrive** en 'OK' om een venster van de ObjectPAL-Editor te openen. (**canArrive** is een ingebouwde methode die wordt uitgevoerd als ObjectPAL toestemming vraagt om een object actief te maken, zoals u op een deur klopt voordat u ergens naar binnen gaat.)
6. Typ de volgende tekst in het venster van de ObjectPAL-Editor:

```
method canArrive(var eventInfo MoveEvent)
self.color = DarkGray
message(self.name, "   canArrive")
sleep(1000)
endmethod
```

   Er staan drie spaties tussen de dubbele aanhalingstekens en het woord *canArrive*, zodat het bericht leesbaar is. U kunt net zoveel spaties gebruiken als u wilt (of geen spaties).
7. Sluit het venster en sla de veranderingen in de methode op.
8. Volg de vorige stappen (u kunt het Klembord gebruiken om u wat typewerk te besparen) om de methodes **arrive**, **setFocus**, **canDepart**, **removeFocus** en **depart** voor *linkerBuitenKader* te bewerken, zoals in de volgende methodevoorbeelden. (Dit zijn ingebouwde methodes, die worden besproken in Appendix B.)

### *arrive-methode*

De ingebouwde **arrive**-methode van een object wordt uitgevoerd als het object actief wordt.

```
method arrive(var eventInfo MoveEvent)
self.color = Gray
message(self.name, "   arrive")
sleep(1000)
endmethod
```

### setFocus-methode

De ingebouwde **setFocus**-methode wordt uitgevoerd als het object *focus* heeft. Dat wil zeggen: als het object invoer van het toetsenbord of de muis kan ontvangen.

```
method setFocus(var eventInfo Event)
self.color = Blue
message(self.name, "   setFocus")
sleep(1000)
endmethod
```

### canDepart-methode

De ingebouwde **canDepart**-methode wordt uitgevoerd als ObjectPAL toestemming vraagt om het object te verlaten.

```
method canDepart(var eventInfo MoveEvent)
self.color = LightBlue
message(self.name, "   canDepart")
sleep(1000)
endmethod
```

### removeFocus-methode

De ingebouwde **removeFocus**-methode wordt uitgevoerd als de focus wordt verwijderd van het object waardoor het geen invoer van het toetsenbord of de muis meer kan ontvangen.

```
method removeFocus(var eventInfo Event)
self.color = DarkGreen
message(self.name, "   removeFocus")
sleep(1000)
endmethod
```

### depart-methode

De ingebouwde **depart**-methode wordt uitgevoerd als het object niet meer actief is.

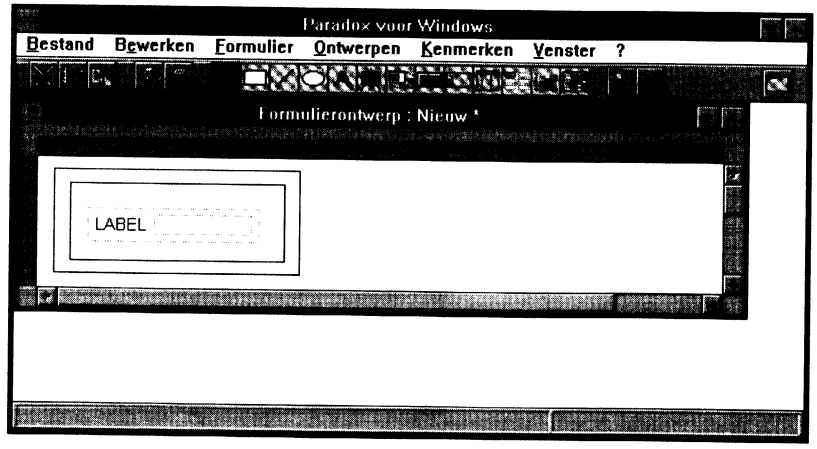
```
method depart(var eventInfo MoveEvent)
self.color = Green
message(self.name, "   depart")
sleep(1000)
endmethod
```

*Stap II: een object met methodes kopiëren*

De volgende taak bestaat uit het kopiëren van het kader met de methodes. Als u *linkerBuitenKader* kopieert, kopieert u ook alle methodes.

1. Selecteer *linkerBuitenKader* en kies vervolgens 'Ontwerpen | Dupliceren' om het kader te kopiëren.
2. Verander de naam van dit gekopieerde kader in *linkerBinnenKader*.
3. Maak *linkerBinnenKader* kleiner. Plaats dit kader vervolgens geheel in *linkerBuitenKader*.
4. Teken met het Veld-hulpmiddel een veldobject in *linkerBinnen-Kader*, zoals in Afbeelding 13-4. Laat het veldobject ongedefinieerd.

Afbeelding 13-4 Een veld tekenen in *linkerBinnenKader*




5. Selecteer *linkerBuitenKader* (en daarmee ook *linkerBinnenKader* en het ongedefinieerde veldobject, want *linkerBuitenKader* sluit *linkerBinnenkader* en het veldobject in) en kies vervolgens 'Ontwerpen | Dupliceren' om deze objecten en de bijbehorende methodes te kopiëren.

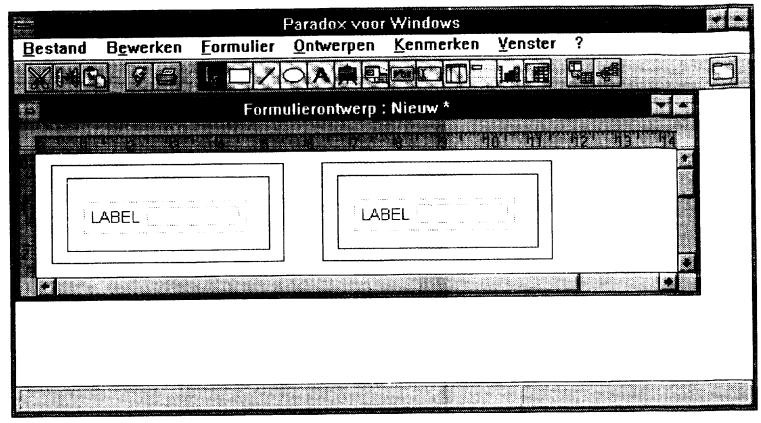
Denk eens na over wat u nu hebt gedaan. U selecteerde één object en daardoor hebt u alle objecten geselecteerd die zich in dat object bevinden. U dupliceerde één object en daardoor hebt u alle objecten gedupliceerd die zich in dat object bevinden, inclusief alle methodes. U hebt het feit dat objecten andere objecten kunnen insluiten, gebruikt om de ontwerpprocedure te stroomlijnen.

*Stap III: gedupliceerde objecten verplaatsen en gebruiken*

Nu moeten de de gedupliceerde objecten worden verplaatst en één actie worden geactiveerd die een reeks acties start.

1. Verplaats de kopieën naar rechts, zoals in Afbeelding 13-5. Geef het buitenste kader de naam *rechterBuitenKader* en geef het binnenste kader de naam *rechterBinnenKader*.

Afbeelding 13-5 De duplicaten verplaatsen




2. Sla dit formulier op en kies vervolgens 'Formulier | Gegevens tonen'.
3. Druk op *Tab* en kijk wat er gebeurt. Druk nogmaals op *Tab*. Als u op *Tab* drukt, activeert u één actie die een kettingreactie start. De kaderkleuren veranderen terwijl elk object reageert op de actiereeks van **canArrive** tot **depart**.

***Opmerking***

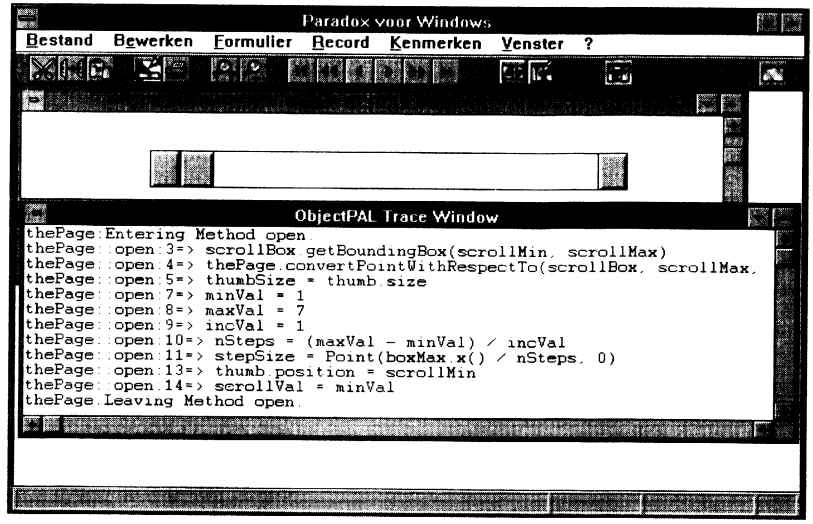
De twee veldobjecten reageren op dezelfde actiereeks door hun ingebouwde methodes uit te voeren. Er verschijnen echter geen berichten, omdat u geen eigen code hebt gekoppeld aan de ingebouwde methodes.

## ObjectPAL-Tracer

U kunt de ObjectPAL-Tracer gebruiken om de ObjectPAL-instructies te laten weergeven terwijl deze worden uitgevoerd. Open in het ontwerpvenster een Editor-venster en kies 'Debug | Uitvoering volgen'. Als u vervolgens overschakelt naar de optie 'Gegevens tonen', geeft het venster van de ObjectPAL-Tracer elke regel ObjectPAL-code weer als die wordt uitgevoerd, zodat u kunt zien wat er gebeurt en wanneer. In Afbeelding 13-6 ziet u de ObjectPAL-Tracer waarin de ObjectPAL-instructies worden weergegeven terwijl deze worden uitgevoerd.

Als u nog meer informatie wilt, kiest u 'Debug | Ingebouwde methodes volgen' en selecteert u de ingebouwde methodes die u wilt volgen tijdens de uitvoering, ongeacht of er aan deze methodes ObjectPAL-code is gekoppeld. In Hoofdstuk 10 vindt u meer informatie over 'Uitvoering volgen' en 'Ingebouwde methodes volgen'.

Afbeelding 13-6 De ObjectPAL-Tracer geeft een overzicht van de ObjectPAL-instructies terwijl deze worden uitgevoerd.




*Samenvatting van de demonstratie*

Deze eenvoudige oefening illustreert enkele belangrijke elementen van het programmeren met ObjectPAL:

- Objecten reageren op acties.
- Eén actie kan een actiereeks activeren die op de achtergrond wordt uitgevoerd (en doet dat gewoonlijk ook).
- De positie in de hiërarchie van ingesloten objecten bepaalt wanneer een object acties ontvangt.
- U kunt deze acties onderscheppen.
- U kunt bepalen hoe objecten reageren.

***Belangrijk***

Raadpleeg Hoofdstuk 12 voor belangrijke informatie over de relaties tussen acties, methodes en insluitende objecten.

## Handelingen en UIObjecten

In deze paragraaf wordt uitgelegd hoe u met ObjectPAL UIObjecten handelingen laat initiëren en op handelingen laat reageren. Eerst wordt algemene informatie gegeven over handelingen en daarna komen de volgende onderwerpen aan de orde:

- De **action**-methode, die voor het UIObject-type is gedefinieerd, waarmee handelingen worden geïnitieerd
- De ingebouwde **action**-methode, waarmee op handelingen wordt gereageerd

## Handelingen

Pas als u handelingen goed begrijpt, kunt u ObjectPAL optimaal benutten.

In Tabel 13-2 worden de vijf basiscategorieën van handelingen beschreven.

Tabel 13-2 Categorieën handelingen

| Categorie | Beschrijving |
|---|---|
| Data-handelingen | Met Data-handelingen kunt u door een tabel sturen en ondermeer records vergrendelen en doorvoeren.De stuurknoppen op de TurboBalk en veel opties in de menu's 'Formulier' en 'Record' activeren deze handelingen. Het formulier kan deze handelingen zelf gebruiken om te proberen records te vergrendelen of door te voeren. |
| Edit-handelingen | De meeste Edit-handelingen wijzigen gegevens in een veld en werken in combinatie met Select-handelingen. U wilt bijvoorbeeld een woord selecteren dat in een veldobject wordt weergegeven (een Select-handeling), en het woord vervolgens naar het Klembord kopiëren (een Edit-handeling). In veldweergave werken Edit-handelingen op tekens in het veldobject. Buiten veldweergave werken deze handelingen op het hele veldobject. Edit-handelingen hebben alleen effect als een object actief (geselecteerd) is. |
| Move-handelingen | Move-handelingen hebben betrekking op de verplaatsing in een veldobject of tussen objecten. Move-handelingen worden meestal door de pijltoetsen geactiveerd. In veldweergave zorgen deze handelingen voor verplaatsing binnen het veld (bijvoorbeeld naar het begin van een regel). Buiten veldweergave zorgen deze handelingen voor verplaatsing naar een ander object. Uiteraard hebben niet alle handelingen betekenis voor alle objecten, maar de handelingen die wel betekenis hebben, worden op de juiste manier uitgevoerd. |
| Field-handelingen | Field-handelingen zijn een speciale categorie Move-handelingen die verplaatsing tussen veldobjecten mogelijk maken (bijvoorbeeld de verplaatsing naar het volgende veldobject in de tabvolgorde). |
| Select-handelingen | Select-handelingen zijn gelijk aan Move-handelingen, alleen breiden deze handelingen een selectie uit tijdens de verplaatsing. U kunt een selectie niet over objecten uitbreiden, maar alleen binnen een veldobject. |

### *Handelingsconstanten*

ObjectPAL kent constanten die u kunt gebruiken om met handelingen te werken. Als u ObjectPAL-code schrijft voor handelingen en UIObjecten, zijn deze constanten zeer belangrijk. Net als de handelingen zelf worden deze constanten in categorieën opgedeeld:

ActionDataCommands
ActionEditCommands
ActionFieldCommands
ActionMoveCommands
ActionSelectCommands

De constanten kunnen online worden weergegeven. Omdat deze constanten zo belangrijk zijn, moet u er minstens eenmaal naar kijken om te weten welke constanten beschikbaar zijn. Als u de lijst wilt zien, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten' om het dialoogvenster 'Constanten' te openen. Kies vervolgens in het paneel 'Types constanten' een van de categorieën. De constanten verschijnen dan in het paneel 'Constanten'.

Raadpleeg de online ObjectPAL Help voor informatie over de betekenis van alle constanten.

*Handelingen, ObjectPAL en Paradox*

Handelingen, ObjectPAL en Paradox zijn nauw met elkaar verbonden, omdat u de meeste taken die u met ObjectPAL kunt uitvoeren of kunt verrichten door Paradox interactief te gebruiken, als handelingen kunt beschouwen. Als u ObjectPAL bijvoorbeeld gebruikt om de waarde van een veld te veranderen, is dat een specifieke handeling. Als u een optie kiest in een menu, is dat weer een andere handeling. Sommige methodes en procedures in de runtime bibliotheek van ObjectPAL zijn eigenlijk verzoeken om een handeling.

Als u ObjectPAL gebruikt, kunt u al deze handelingen initiëren en bovendien kunt u opgeven hoe objecten op deze handelingen reageren. In de volgende paragrafen wordt uitgelegd hoe u dat doet.

## Handelingen initiëren

Als u een handeling wilt initiëren—dat wil zeggen een UIObject wilt opdragen iets te doen—gebruikt u de **action**-methode die voor het UIObject-type is gedefinieerd. De volgende instructie gaat bijvoorbeeld naar het volgende record in een tabelframe of multi-record object dat met de tabel *Order* is verbonden:

```
ORDER.action(DataNextRecord)
```

Deze instructie gebruikt de constante DataNextRecord om aan te geven welke handeling moet worden uitgevoerd.

De volgende instructie roept het ingebouwde dialoogvenster 'Zoeken en vervangen' van Paradox aan:

```
order.action(DataSearchReplace)
```

De gebruiker kan waarden in het dialoogvenster typen en Paradox voert dan de zoek- en vervanghandeling uit. U hoeft het wiel niet opnieuw uit te vinden.

***Opmerking***

Sommige methodes en procedures in de runtime bibliotheek van ObjectPAL zijn eigenlijk aanroepen van de **action**-methode. De eerste instructie is bijvoorbeeld een aanroep van de tweede instructie.

```
tabelFrame.nextRecord()

tabelFrame.action(DataNextRecord)
```

De twee instructies zijn identiek en u kunt deze beide in uw code gebruiken.

*De action-methode en het actiemodel*

Als code die aan een ontwerpobject is gekoppeld, **action** aanroept, wordt een ActionEvent gegenereerd die door Paradox wordt behandeld volgens de definitie van het actiemodel (beschreven in Hoofdstuk 12): de actie gaat eerst naar het formulier en activeert de ingebouwde **action**-methode. Het formulier stuurt de actie standaard naar de ingebouwde **action**-methode van het beoogde doelobject en van daaruit kan de actie omhoogborrelen in de hiërarchie van ingesloten objecten, totdat een object de actie behandelt of totdat het formulier voor de tweede keer wordt bereikt.

In de vorige voorbeelden is een objectnaam gebruikt met **action** om een doelobject op te geven en meestal is dat de juiste werkwijze. Als u naar het laatste record van het tabelframe *Order* wilt gaan, gebruikt u de volgende code:

```
order.action(DataEnd)
```

In dit geval gaat de actie naar het formulier dat de actie direct naar *Order* stuurt.

*Action aanroepen met een object*

In gevallen waarin u meer algemene code wilt, kunt u **action** aanroepen met een objectvariabele of zonder een object op te geven. De reeks acties en objecten die daaruit resulteren, is afhankelijk van de hiërarchie van ingesloten objecten. Deze reeks is met name afhankelijk van de relatie tussen het aanroepend object (het object waarvan de code **action** heeft aangeroepen) en het actieve object.

*Action aanroepen zonder een object*

Als de **action**-methode van het UIObject-type wordt aangeroepen zonder een object, heeft dit hetzelfde effect als wanneer u **action** aanroept en *self* opgeeft. Met andere woorden: de volgende instructies zijn identiek:

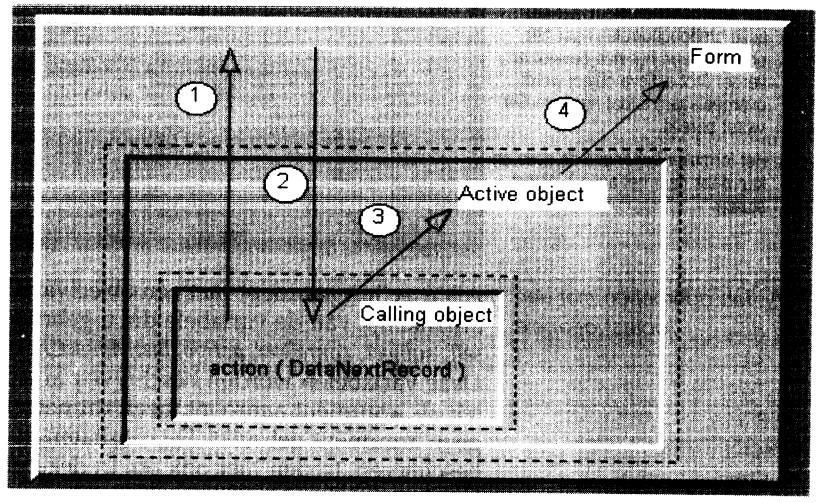
```
action(DataNextRecord)
self.action(DataNextRecord)
```

Afbeelding 13-7 en Afbeelding 13-8 tonen twee scenario's om **action** aan te roepen zonder een object op te geven. In het eerste scenario sluit het actieve object het aanroepend object in. In het tweede scenario bevinden het actieve object en het aanroepend object zich in aparte hiërarchieën van ingesloten objecten.

In Afbeelding 13-7 sluit het actieve object het aanroepend object in (stippellijnen vertegenwoordigen mogelijke tussenliggende insluitende objecten). Als het aanroepend object **action** aanroept met een constante (bijvoorbeeld DataNextRecord), wordt een ActionEvent gegenereerd die de ingebouwde **action**-methode voor het formulier activeert. Het formulier stuurt de actie standaard naar de ingebouwde **action**-methode van het aanroepend object, dat de actie omhoog laat borrelen naar het insluitende object en zo verder totdat

het actieve object wordt bereikt. Als de handeling voor het actieve object betekenis heeft (DataNextRecord heeft bijvoorbeeld betekenis voor een tabelframe), voert het actieve object de handeling uit. Als de handeling geen betekenis heeft, laat het actieve object de actie omhoog borrelen naar het insluitende object en zo verder totdat het formulier voor de tweede keer wordt bereikt en het formulier de actie behandelt.

Afbeelding 13-7 Actief object sluit aanroepend object in

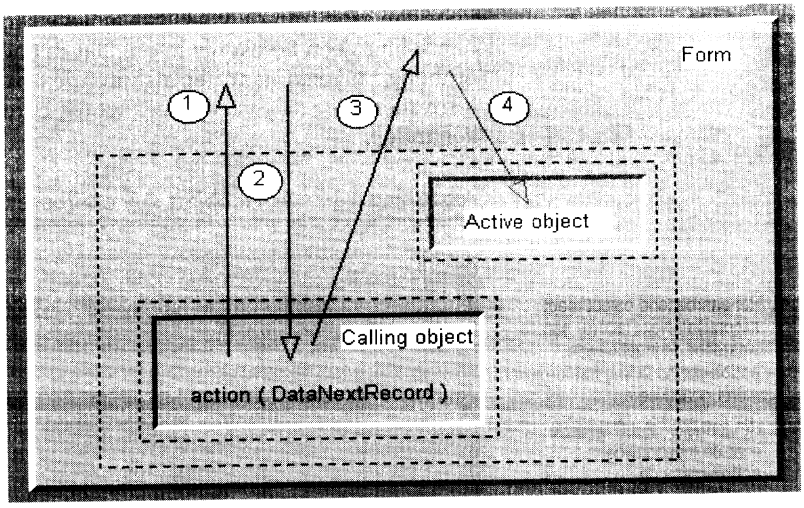



1. Het aanroepend object roept action(DataNextRecord) aan, waardoor de ingebouwde **action**-methode van het formulier wordt geactiveerd.
2. Het formulier stuurt de actie naar de ingebouwde **action**-methode van het aanroepend object.
3. Het aanroepend object laat de actie omhoogborrelen naar het insluitend object en verder totdat de actie bij het actieve object aankomt.
4. Het actieve object voert de handeling uit als het dat kan of laat de actie omhoogborrelen als het dat niet kan.

In Afbeelding 13-8 bevindt het aanroepend object zich niet in het actieve object. Deze objecten bevinden zich in aparte hiërarchieën van ingesloten objecten. In dit geval gaat de ActionEvent naar het formulier, dan terug naar het aanroepend object en vervolgens borrelt de ActionEvent omhoog in de hiërarchie van ingesloten objecten, waarin zich het actieve object *niet* bevindt, totdat het formulier opnieuw wordt bereikt. Het actieve object ziet de ActionEvent niet terwijl deze omhoogborrelt naar het formulier, maar als het formulier de actie voor de tweede keer ontvangt, kan het deze naar het actieve object sturen. (Dezelfde volgorde vindt plaats als het actieve object zich in het aanroepend object bevindt.)

Afbeelding 13-8 Aparte hiërarchieën, object niet opgegeven

1. Het aanroepend object roept action(DataNextRecord)aan, waardoor de ingebouwde **action**-methode van het formulier wordt geactiveerd.
2. Het formulier stuurt de actie naar de ingebouwde **action**-methode van het aanroepend object.
3. Het aanroepend object laat de actie omhoogborrelen naar het object waarin het zich bevindt en verder (het actieve object wordt overgeslagen) totdat het formulier wordt bereikt.
4. Het formulier behandelt de actie en kan deze naar het actieve object sturen.




*Action aanroepen met een objectvariabele*

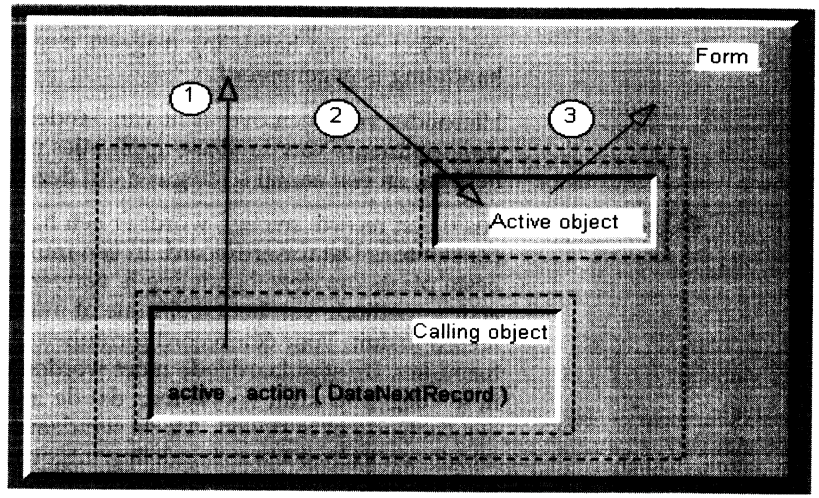
Als u **action** aanroept met een objectvariabele, zijn de resultaten afhankelijk van de variabele die u gebruikt. (De objectvariabelen zijn *self, subject, container, active, lastMouseClicked* en *lastMouseRightClicked*. Deze variabelen worden beschreven in Appendix B.) Als u **action** met *self* als argument aanroept, heeft dit hetzelfde effect als wanneer u geen argument gebruikt. De resultaten die in de vorige paragraaf zijn beschreven, treden hier ook op. In deze paragraaf worden de effecten beschreven van de aanroep van **action** met *active* als argument. Dezelfde principes zijn van toepassing als de andere objectvariabelen worden gebruikt.

In Afbeelding 13-9 wordt het effect getoond van de volgende instructie:

```
active.action(DataNextRecord)
```

Deze instructie genereert een ActionEvent die de ingebouwde **action**-methode van het formulier activeert. Het formulier stuurt de actie naar de ingebouwde **action**-methode van het actieve object. Het actieve object voert de handeling uit als dat mogelijk is en laat deze omhoogborrelen als dat niet mogelijk is.

Afbeelding 13-9 Aparte hiërarchieën, actief object opgegeven




1. Het aanroepend object roept active.action(DataNextRecord) aan, waardoor de ingebouwde **action**-methode van het formulier wordt geactiveerd.
2. Het formulier stuurt de actie naar de ingebouwde **action**-methode van het actieve object.
3. Het actieve object voert de handeling uit als het dat kan of laat deze omhoogborrelen als het dat niet kan.

Met ObjectPAL kunt u dus zo specifiek of zo algemeen zijn als u wilt bij een aanroep van de **action**-methode van het UIObject-type. De effecten van algemene code zijn afhankelijk van de positie van het aanroepend object en het actieve object in de hiërarchie van ingesloten objecten.

## Reageren op handelingen

Elk UIObject (inclusief het formulier) heeft een ingebouwde **action**-methode die bepaalt hoe het object reageert op handelingen. U kunt code aan de ingebouwde **action**-methode van een object koppelen om te bepalen hoe dat object in specifieke situaties reageert.

Het *eventInfo*-argument dat door de ingebouwde **action**-methode van een object wordt gebruikt, bevat informatie over de bedoelde handeling: de **id**-methode die voor het ActionEvent-type is gedefinieerd, geeft deze informatie terug. Stel dat de volgende code is gekoppeld aan de ingebouwde **action**-methode van een recordobject:

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataUnlockRecord then ; als handeling record probeert te
                                             ; ontgrendelen
      controleerRecord() ; voer een eigen procedure uit
   endIf
endMethod
```

Deze code roept **id** aan om te testen op *eventInfo*. Als **id** een waarde teruggeeft die gelijk is aan de constante DataUnlockRecord, roept de volgende instructie een eigen procedure aan om te controleren of de recordwaarden geldig zijn.

**Belangrijk** Een ingebouwde **action**-methode van een object wordt uitgevoerd in reactie op alle handelingen, ongeacht of de handeling is gegenereerd

door ObjectPAL, door Paradox of door de gebruiker die interactief met het formulier werkt. Daarom is de ingebouwde **action**-methode een uitstekende plaats om uw code te plaatsen als u wilt bepalen hoe een object op een handeling reageert, ongeacht de manier waarop de handeling is gegenereerd.

Hieronder volgt een overzicht van handelingen waarover programmeurs van database-applicaties met name controle willen hebben, en een aantal strategieën om deze controle te verkrijgen.

*Record invoegen*

Als u een record invoegt, wordt er een handeling geactiveerd. De constante is DataInsertRecord. In een multi-record formulier is de beste plaats om deze handeling te verwerken de ingebouwde **action**-methode van het tabelframe of multi-record object dat het record insluit. Voor één-record formulieren koppelt u code aan het formulier. De standaardcode moet worden uitgevoerd om het record in te voegen. U kunt voorkomen dat de invoeging plaatsvindt door een foutcode in te stellen. Als de standaardcode wordt uitgevoerd, wordt een RefreshMove-handeling uitgevoerd naar het tabelframe of multi-record object (de ingebouwde **arrive**-methode en **depart**-methode van het recordobject worden geactiveerd), wordt een nieuw record ingevoegd, wordt het scherm bijgewerkt en wordt het juiste veld in het nieuwe record gemarkeerd. U kunt standaardwaarden instellen voor velden, de kleur van het record veranderen enzovoort.

Als een record *om welke reden dan ook* wordt ingevoegd, is DataInsertRecord de enige handeling die u hoeft te onderscheppen om te bepalen hoe nieuwe records worden behandeld.

De code in het volgende voorbeeld kijkt of er een poging wordt gedaan om een record in te voegen. Deze code roept **doDefault** aan om de standaardcode uit te voeren en stelt vervolgens de waarde van het veldobject *Bedrijfsnaam* in op Borland.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataInsertRecord then
      doDefault
      Bedrijfsnaam.value = "Borland"
   endIf
endmethod
```

*Record ontgrendelen en doorvoeren*

Als u wijzigingen in gegevens op recordniveau wilt behandelen, moet u reageren op twee handelingen, DataUnlockRecord en DataPostRecord. Beide handelingen kunnen worden geïnitieerd door een gebruiker die interactief met het formulier werkt of door een ObjectPAL-instructie. Toch zijn het verschillende handelingen.

Een DataUnlockRecord-handeling zegt in feite "Ik ben klaar met dit record. Ontgrendel het en voer de veranderingen in de tabel door. Als de veranderingen ervoor zorgen dat het record wegvliegt, laat het dan gaan." Een DataPostRecord-handeling zegt echter "Voer deze veranderingen in de tabel door. Ik wil bij het record blijven. Houd het

record vergrendeld en als het wegvliegt, ga dan mee." Zoals u ziet, zijn het twee verschillende handelingen, die u dus beide moet onderscheppen. U kunt elke handeling blokkeren door een foutcode in te stellen voordat de standaardcode wordt uitgevoerd.

Net als bij DataInsertRecord initieert de standaardcode van deze handelingen een RefreshMove-handeling naar het tabelframe of het multi-record object. Vervolgens wordt de juiste databasebewerking uitgevoerd (ontgrendelen of doorvoeren) en wordt teruggekeerd naar het juiste veld in het nieuwe huidige record.

Deze verplaatsing in drie stappen (naar het tabelframe of multi-record object gaan, de bewerking uitvoeren en vervolgens naar het record terugkeren) is nodig bij records die wegvliegen als een sleutelwaarde wordt veranderd. Bovendien worden de ingebouwde **arrive**- en **depart**-methodes van het record geactiveerd, waardoor u desgewenst de gebruikersinterface kunt bijwerken.

De code in het volgende voorbeeld is gekoppeld aan het tabelframe. De code reageert op een DataUnlockRecord- en DataPostRecord-handeling door op de waarde van het veldobject *Aantal* te testen. Als de waarde van *Aantal* kleiner is dan de waarde van *minWaarde* (een constante die elders is gedefinieerd), stelt deze code de foutcode in om de handeling te blokkeren en verschijnt er een bericht om de gebruiker te informeren.

```
method action(var eventInfo ActionEvent)
   var
      idWaarde SmallInt
   endVar

   idWaarde = eventInfo.id() ; idWaarde wordt gebruikt om typewerk te besparen

   if idWaarde = DataUnlockRecord or idWaarde = DataPostRecord then
      if Aantal.value < minWaarde then
         message("Voer een hoger aantal in.")
         eventInfo.setErrorCode(UserError)
      endIf
   endIf
endmethod
```

*Record verwijderen*

Als u een record verwijdert, wordt er een handeling geactiveerd. De constante is DataDeleteRecord. Net als in de vorige voorbeelden gebeurt er pas iets met het record als de standaardcode wordt uitgevoerd. Een foutcode kan dit voorkomen. Als de standaardcode wordt uitgevoerd, veroorzaakt deze een verplaatsing naar het tabelframe of het multi-record object. Vervolgens wordt de database gevraagd het record te verwijderen en ten slotte zorgt de code ervoor dat wordt teruggekeerd naar het nieuwe huidige record.

Als u een record verwijdert, wordt deze handeling altijd gegenereerd. DataDeleteRecord is dus de handeling die u moet onderscheppen om verwijderingen van records te behandelen.

Het volgende voorbeeld reageert op DataDeleteRecord door een dialoogvenster te openen waarin de gebruiker het besluit om een record te verwijderen kan bevestigen.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataDeleteRecord then
      if msgQuestion ("Verwijderen?", "Dit record verwijderen?") = "Yes" then
         doDefault
      else
         eventInfo.setErrorCode(UserError)
      endIf
   endif
endmethod
```

*Uizondering tijdens opfrissen*

Als iemand in het netwerk (of in een ander venster) een record wijzigt in het bereik dat u in uw formulier ziet, vindt er een handeling plaats. De constante is DataRefresh. Tijdens de uitvoering zorgt de standaardcode voor een verplaatsing naar het tabelframe of het multi-record object, wordt de huidige positie van het record herberekend en wordt teruggekeerd naar het veld dat actief was (in een poging de bewerktoestand te behouden).

Als gegevens worden opgefrist in een record dat op dit moment niet in uw formulier wordt weergegeven, is de handeling DataRefreshOutside. DataRefresh en DataRefreshOutside worden gebruikt in applicaties die in een netwerk worden gebruikt. U kunt op deze handelingen reageren door een bericht weer te geven waarmee de gebruiker wordt geïnformeerd, zoals in het volgende voorbeeld.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataRefresh or eventInfo = DataRefreshOutside then
      message("Gegevens opfrissen...")
   endif
endmethod
```

*Aankomen bij records en records verlaten*

U kunt één handelingsconstante, namelijk DataArriveRecord, gebruiken om te reageren op een handeling die op verschillende manieren kan zijn geïnitialiseerd. Een DataArriveRecord-handeling vindt plaats als het formulier "aankomt bij" (de inhoud toont van) een nieuw of ander record. Als u bijvoorbeeld naar het volgende of vorige record gaat, wordt een DataArriveRecord-handeling geïnitieerd. Dit gebeurt ook als u een record invoegt, verwijdert of bewerkt. DataArriveRecord is een nuttige, algemene handeling.

Raadpleeg *Leren werken met ObjectPAL* voor een voorbeeld van het gebruik van DataArriveRecord.

Zoals in de vorige voorbeelden is uitgelegd, worden de ingebouwde **arrive**- en **depart**-methodes van een recordobject geactiveerd als neveneffecten van de standaardcode als u een record invoegt, verwijdert, ontgrendelt of doorvoert. Dit betekent dat als u bijvoorbeeld de kleur van een record wilt instellen, u code kunt koppelen aan de ingebouwde **arrive**-methode. U weet dan zeker dat deze code

wordt uitgevoerd voor alle elementaire handelingen. Als u derhalve iets wilt doen met de gebruikersinterface telkens wanneer u aankomt bij een record of een record verlaat (ongeacht of het gaat om een invoeging, verwijdering enzovoort), er is altijd een plaats om uw code te koppelen. Als u bijvoorbeeld op een ander record klikt, wordt er geen handeling gegenereerd als het record niet is vergrendeld. U moet vertrouwen op de **depart**- en **arrive**-reeks.

### Waar plaats ik mijn code?

Als u handelingen wilt behandelen, is de beste plaats om code te koppelen de ingebouwde **action**-methode van het "kleinste gemene insluitend object". Dat wil zeggen: het insluitend object op het laagste niveau van de objecten waarin u bent geïnteresseerd. Stel dat u een tabelframe hebt en iets speciaals wilt doen met een veldobject in dat frame. Het tabelframe sluit per definitie de veldobjecten in, maar dat doen de pagina en het formulier ook. Het formulier staat "bovenaan" in de hiërarchie van ingesloten objecten en staat dus het verst af van de veldobjecten. Het tabelframe is het kleinste gemene insluitend object. Plaats uw code dus in het tabelframe.

In één-record formulieren (die geen recordobject insluiten) is de beste werkwijze algemene code aan de pagina te koppelen en code die specifiek voor het object geldt, aan dat object te koppelen. U kunt ook de veldobjecten groeperen en code koppelen aan de ingebouwde **action**-methode van het groepsobject of een kader tekenen rond de veldobjecten en code koppelen aan de ingebouwde **action**-methode van het kader.

***Belangrijk*** Hoewel de meeste handelingen helemaal terugborrelen naar het formulier en op het niveau van het formulier kunnen worden verwerkt, is het beter de intelligentie in te kapselen in het betrokken object.

U ziet de voordelen van inkapseling als u de eerste keer groepen objecten knipt en plakt tussen formulieren. U kunt natuurlijk alles op het formulierniveau plaatsen, met name als u **isPreFilter** gebruikt, maar u maakt dan binnen de korst mogelijke tijd uiterst gecompliceerde formuliermethodes.

*Handelingen per categorie filteren*

Zoals u in de vorige voorbeelden hebt gezien, kunt u **id** in combinatie met constanten gebruiken om op specifieke handelingen te reageren. Stel echter dat u een bepaalde categorie handeling wilt behandelen en de rest aan Paradox wilt overlaten. Als u wilt weten tot welke categorie een handeling behoort, roept u de **actionClass**-methode van het ActionEvent-type aan en vergelijkt u de teruggegeven waarde met een van de ActionClasses-constanten: DataAction, EditAction, FieldAction, MoveAction of SelectAction. Stel dat u alle DataAction-constanten zelf wilt behandelen, maar Paradox alle andere handelingen wilt laten verwerken. U kunt dan bijvoorbeeld de volgende code schrijven:

```
method action(var eventInfo ActionEvent)
   var
      deKlasse SmallInt
   endVar
   deKlasse = eventInfo.actionClass()
   if deKlasse = DataAction then
      disableDefault
      ikDoeHetWel(eventInfo)     ; geef de actie aan een eigen procedure door
   endIf                         ; sta anders standaardbehandeling toe
endmethod
```

Deze code roept **actionClass** aan en slaat de teruggegeven waarde op in een variabele met de naam *deKlasse*. Als de waarde van *deKlasse* gelijk is aan de waarde van de constante DataAction, geeft een volgende instructie *eventInfo* door aan een eigen procedure met de naam **ikDoeHetWel**. Deze procedure verwerkt alle handelingen in de betreffende categorie.

## Voorbeeld: handelingen en tabelframes

Het UIObject-type kent verschillende constanten en methodes voor het werken met tabelframe-objecten. Stel dat een formulier een tabelframe bevat dat met de tabel *Order* is verbonden. U kunt de **action**-methode in combinatie met constanten gebruiken om de invoegpositie te verplaatsen en gegevens in het tabelframe te bewerken, zoals u in de volgende instructies ziet.

```
Order.action(DataNextRecord)    ; gaat naar het volgende record
Order.action(DataPriorRecord)   ; gaat naar het vorige record
Order.action(DataInsertRecord) ; voegt een record in het tabelframe in
Order.action(DataBeginEdit)     ; zet het tabelframe in de bewerkmodus
Order.action(DataLockRecord)    ; vergrendelt het huidige record
Order.action(DataEndEdit)       ; beëindigt de bewerkmodus, voert veranderingen
                                ; in het huidige record door
```

U kunt veel meer constanten gebruiken in combinatie met **action**. Raadpleeg de online ObjectPAL Help voor informatie.

De runtime bibliotheek van ObjectPAL bevat onder andere de volgende methodes:

- **attach**
- **locate**
- **nRecords**

Deze en aanverwante methodes werken op dezelfde manier als hun tegenhangers in het type TCursor. Het enige verschil is dat de resultaten in een tabelframe worden weergegeven. U kunt tabelframes en TCursors samen gebruiken. Met de TCursor werkt u met gegevens op de achtergrond. Daarna, als de verwerking voltooid is, gebruikt u het tabelframe om de resultaten weer te geven. Raadpleeg Hoofdstuk 16 voor voorbeelden.

## Voorbeeld: handelingen en records

De TurboBalk heeft geen echt hulpmiddel om een recordobject te maken, maar u kunt op de volgende manieren records in de gebruikersinterface manipuleren:

- Gebruik de handelingsconstante DataArriveRecord om alle handelingen te vinden die ervoor zorgen dat de tabellen uit het gegevensmodel naar een ander record wijzen. Telkens wanneer u een record verschuift, invoegt, verwijdert of doorvoert, met de muis naar een record gaat of wanneer een zichtbaar record in een netwerk wordt gewijzigd, wordt een DataArriveRecord-handeling gegenereerd.

  Deze handeling verschilt van andere gebruikelijke handelingen, omdat u deze niet kunt blokkeren. DataArriveRecord dient ter kennisgeving: de handeling heeft al plaatsgevonden.

  Stel dat u een formulier hebt voor de verwerking van bestelinformatie en dat u alleen het veldobject *Rekeningnummer* wilt laten weergeven voor bestellingen op rekening. U controleert hiervoor de waarde van het veldobject *Betaalwijze* voor elk record: als de waarde "Rekening" is, geeft u het veldobject *Rekeningnummer* weer. Een mogelijke aanpak (niet aan te bevelen) is een gigantische **switch**-instructie te schrijven die alle mogelijke handelingen die met records zijn verbonden, opvangt (DataNextRecord, DataPriorRecord, DataPostRecord, DataInsertRecord, DataToggleEdit enzovoort). De aanbevolen manier (zie het volgende voorbeeld) is DataArriveRecord in de ingebouwde **action**-methode van het formulier te gebruiken.

```
method action(var eventInfo ActionEvent)
if eventInfo.isPreFilter()
   then
   ; deze code wordt voor elk object op het formulier uitgevoerd
   else
   ; deze code wordt alleen voor het formulier zelf uitgevoerd
   if eventInfo.id() = DataArriveRecord then
      if Betaalwijze.value = "Rekening" then
         Rekeningnummer.visible = True
      else
         Rekeningnummer.visible = False
      endIf
   endIf
endIf
endMethod
```

- Maak een multi-record object en definieer vervolgens de record-layout zo dat één record horizontaal en één record verticaal wordt weergegeven (ook 1 x 1 genoemd, uitgesproken als "één-bij-één").

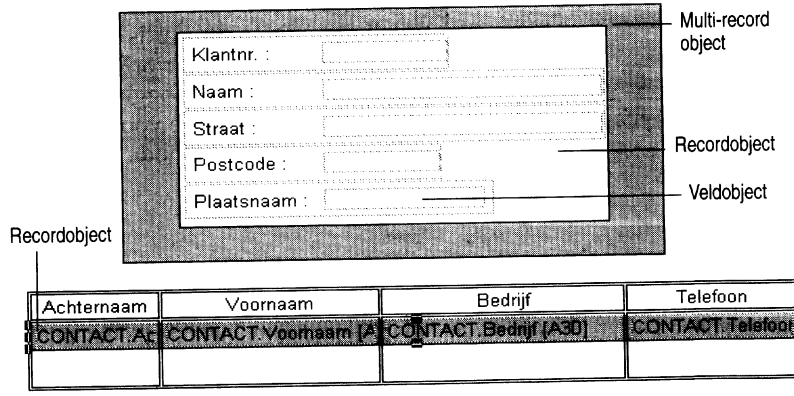
- Gebruik het recordobject dat zich in elk tabelframe bevindt (het Objectenschema biedt eenvoudig toegang).

Bij beide werkwijzen kunt u op dezelfde manier met een record werken als u met andere UIObjecten doet. U kunt bijvoorbeeld een record inspecteren, kenmerken instellen en methodes koppelen. Een overzicht van recordkenmerken is online beschikbaar. Als u de lijst wilt bekijken, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Kenmerken'. Vervolgens kiest u 'Record' in de kolom 'Objecten'. De kenmerken verschijnen in de kolom 'Kenmerken'.

In Afbeelding 13-10 ziet u twee samengestelde objecten: een één-bij-één multi-record object dat is verbonden met de tabel *Klant* en een tabelframe dat is verbonden met de tabel *Contact*. Het donkergrijze gebied van het multi-record object vertegenwoordigt het multi-record object, dat één record bevat, en het lichtgrijze gebied vertegenwoordigt het record zelf. De witte gebieden vertegenwoordigen veldobjecten die zich in het record bevinden. In het tabelframe vertegenwoordigt het grijze gebied het recordobject.

Raadpleeg het *Handboek* voor meer informatie over recordobjecten.

Afbeelding 13-10 Recordobjecten in een multi-record object en een tabelframe




## UIObjecten: snelle manieren en speciale gevallen

In deze paragraaf wordt beschreven hoe u UIObjecten kopieert en maakt, hoe u prototypes van UIObjecten maakt en hoe u berekende velden definieert.

### *Objecten kopiëren*

Als u een object kopieert, krijgt u een exacte kopie van het object, inclusief de kenmerken, methodes en procedures. Alleen de naam van het object verandert. Als u een object kopieert, moet u niet vergeten code bij te werken die met behulp van een naam naar het object verwijst. Als een methode in een object *Self* gebruikt en u het object kopieert, verwijst *Self* naar de kopie en niet naar het

oorspronkelijke object, omdat *Self* altijd verwijst naar het object waaraan de code is gekoppeld.

Er bestaat geen koppeling tussen het oorspronkelijke object en de kopie. Als u een methode verandert die aan een object is gekoppeld, heeft dit geen effect op methodes in het andere object, zoals u in het volgende voorbeeld ziet:

1. Maak een formulier met één knop en wijzig de **pushButton**-methode van de knop als volgt:

```
method pushButton (var eventInfo Event)
   message("Ik ben de oorspronkelijke knop.")
endMethod
```

2. Sluit het venster van de ObjectPAL-Editor, sla de veranderingen op en noem de knop *mijnOrigineel*.

3. Kies 'Ontwerpen | Dupliceren' om deze knop te kopiëren. (U bereikt hetzelfde effect als u het object knipt en plakt of kopieert en plakt.)

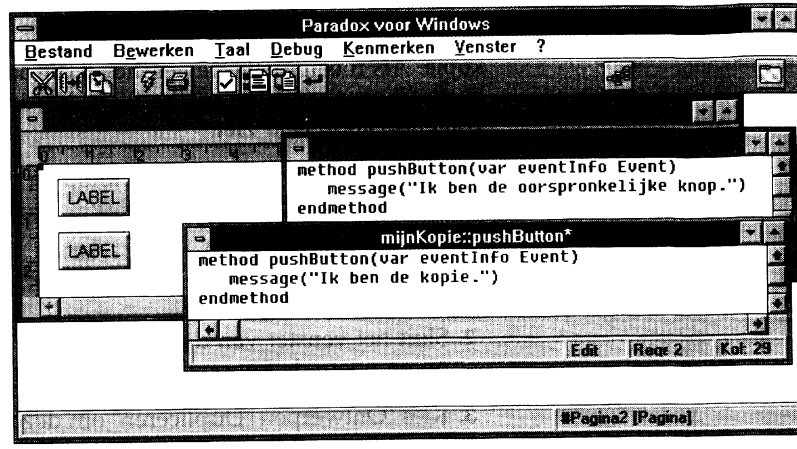
4. Inspecteer de **pushButton**-methode van de nieuwe knop. Deze is hetzelfde. Verander de methode:

```
method pushButton (var eventInfo Event)
   message("Ik ben de kopie.")
endMethod
```

5. Sluit het venster van de ObjectPAL-Editor, sla de veranderingen op en noem de knop *mijnKopie*.

6. De methode van de oorspronkelijke knop wordt niet gewijzigd. Als u dit wilt controleren, kiest u 'Formulier | Gegevens tonen' en klikt u op beide knoppen om de berichten op de statusregel te vergelijken. U kunt ook de methodes zelf vergelijken door elk in een eigen venster van de ObjectPAL-Editor weer te geven, zoals in Afbeelding 13-11.

Afbeelding 13-11 Het origineel en de kopie




### Prototypes maken van objecten

Een snelle manier om objecten te maken die dezelfde kenmerken en methodes hebben, is een prototype van een object te maken. Hieronder volgt een manier om dit te doen:

1. Gebruik Paradox interactief om een object te maken, de kenmerken in te stellen en de methodes te schrijven.
2. Selecteer het object.
3. Kies 'Ontwerpen | Kopiëren naar TurboBalk'.

Nu heeft elk object van dat type die kenmerken en methodes. Als u bijvoorbeeld een veldobject maakt, de kenmerken instelt, methodes schrijft en het object vervolgens naar de TurboBalk kopieert, krijgt *elk* volgend veld die kenmerken en methodes, inclusief velden die u maakt met het Veld-hulpmiddel van de TurboBalk, velden die u maakt met ObjectPAL en velden die als onderdeel van samengestelde objecten zijn gemaakt (bijvoorbeeld de velden in een tabelframe). Zie het *Handboek* voor meer informatie over het maken en opslaan van prototype-objecten.

### UIObjecten maken

U kunt UIObjecten vanuit een methode maken en manipuleren met behulp van **create**, **delete**, **methodGet**, **methodSet** en **methodDelete**. De constanten (zoals BoxTool) voor het maken van objecten kunnen online worden weergegeven. Als u de lijst wilt bekijken, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten' en vervolgens 'UIObjectTypes'. De constanten verschijnen in de kolom 'Constanten'.

**Opmerking** Objecten die in een methode worden gemaakt, zijn onzichtbaar totdat u het kenmerk 'Visible' op True instelt.

Het volgende voorbeeld manipuleert objecten als u 'Gegevens tonen' kiest of een ontwerpvenster opent. Dit voorbeeld maakt een formulier en enkele rechthoeken, wijst aan de rechthoeken methodes toe en verandert de kleuren. (U kunt objecten maken in een ontwerpvenster of in de modus 'Gegevens tonen', maar u kunt alleen in een ontwerpvenster met de methodes werken.)

```
method pushButton(var eventInfo Event)
var
   f form
   ui Array[10] UIObject
   c array[3] LongInt
   i SmallInt
endVar

c[1] = Red
c[2] = Green
c[3] = Blue

f.create() ; Maak een nieuw formulier in een ontwerpvenster

; Maak enkele rechthoeken
for i from 1 to ui.size()
   ui[i].create(BoxTool,i*100,i*100,1000,1000,f)
   ui[i].color = c[i.mod(3)+1]
   ui[i].visible = True
endFor

; Benoem de rechthoeken
for i from 1 to ui.size()
   ui[i].name = "Kadernummer" + String(i)
endFor

f.save("MijnFormulier") ; Sla het formulier op

; Maak een open()-methode voor elk object
for i from 1 to ui.size()
   ui[i].MethodSet("open",
       "method open(var eventInfo Event)
        self.color = Red
        self.visible = True
        Beep()
        endMethod
        ")
endFor
```

Het volgende voorbeeld laat zien hoe u met objecten werkt in een ontwerpvenster:

```
for i from 1 to ui.size()
   ui[i].setPosition(2000 * rand(), 2000 * rand(),
                     2000 * rand(), 2000 * rand())
   ui[i].color = c[mod(rand()*100,3) + 1]
   ui[i].visible = True
endFor
```

Het volgende voorbeeld laat zien hoe u met objecten werkt tijdens de uitvoering:

```
f.run() ; Schakel over naar 'Gegevens tonen'

for i from 1 to ui.size()
```

```
   ui[i].setPosition(6000*rand(), rand()*6000,
                     4000 * rand(), 4000 * rand())
   ui[i].color = c[mod(rand()*100,3) + 1]
   ui[i].visible = True
endFor

f.design() ; keer terug naar een ontwerpvenster

; Verwijder de objecten
for i from 1 to ui.size()
   ui[i].visible = False
   ui[i].delete()
endFor

endmethod
```

## ObjectPAL in berekende velden

In deze paragraaf wordt beschreven hoe u ObjectPAL in berekende velden gebruikt.

De regel voor het gebruik van ObjectPAL in een berekend veld is eenvoudig: u kunt een berekend veld zo definiëren dat het een ObjectPAL-instructie of -uitdrukking gebruikt die één waarde teruggeeft of oplevert.

**Opmerking** In het *Handboek* wordt uitgelegd hoe u een berekend veld maakt en definieert. In deze paragraaf wordt ingegaan op ObjectPAL.

Zoals u in Tabel 13-3 ziet, kan een berekend veld een van de volgende elementen gebruiken:

- Vaste waarden
- Variabelen, op voorwaarde dat deze worden gedeclareerd binnen het bereik van het berekende veld en dat er een waarde aan is toegewezen
- Objectkenmerken
- Elementaire taalonderdelen
- Eigen methodes die aan andere objecten zijn gekoppeld (of aan het veld zelf). U moet eerst een UIObject-variabele declareren binnen het bereik van het berekende veld en een **attach**-instructie gebruiken om de variabele met een UIObject te associëren.
- Alle methodes en procedures in de runtime bibliotheek (RTL: runtime library) van ObjectPAL die een waarde teruggeven (inclusief een logische waarde).
- Speciale functies (zoals **Sum** and **Avg**) die speciaal bedoeld zijn voor gebruik in berekende velden

De DataRecalc-handeling is nauw verbonden met berekende velden, zoals na de volgende tabel wordt beschreven.

Tabel 13-3 ObjectPAL-elementen in berekende velden

| Element | Commentaar |
|---|---|
| 5 | Vaste waarde |
| "a" | Vaste waarde |
| *x* | Variabele. Moet binnen het bereik van het berekende veld zijn gedeclareerd. Aan de variabele moet een waarde zijn toegewezen. (Zie het voorbeeld na deze tabel.) |
| *x* + 5 | Eenvoudige uitdrukking. De regels voor het werken met variabelen zijn van toepassing. |
| self.name | Kenmerk. Geeft de naam van het veld weer (String). |
| hetKader.color | Kenmerk. Geeft de waarde van een geheel getal weer die de kleur van het object vertegenwoordigt. |
| iif(Regio.Value = "NW", 0.075, 0) | Elementair taalonderdeel **iif**. De waarde van het berekende veld hangt af van de waarde van het veldobject 'Regio'. |
| uio.objEigenMethode() | Een eigen methode die is gekoppeld aan een ander object. De eigen methode moet een waarde teruggeven. (Zie het voorbeeld na deze tabel.) |
| tc.open("order.db") | RTL-methode. Het veld geeft True weer als **open** lukt; anders wordt False weergegeven. De TCursor moet binnen het bereik van het veld zijn gedeclareerd. |
| Avg([DUIKSPUL.Prijs]) | Speciale functie. Werkt op het veld 'Prijs' van de tabel *Duikspul*. De tabel moet zich in het gegevensmodel van het formulier bevinden. Aanhalingstekens worden niet gebruikt, spaties zijn toegestaan. |
| tc.cAverage("Prijs") | RTL-methode. De TCursor moet al eerder zijn gedeclareerd en geopend. De tabel hoeft zich niet in het gegevensmodel te bevinden. Als een veldnaam spaties bevat, zijn aanhalingstekens verplicht. (Zie het voorbeeld na deze tabel.) |

### *Voorbeelden van berekende velden*

In deze paragraaf worden de volgende voorbeelden gegeven:

- Een variabele gebruiken
- Een eigen methode aanroepen die is gekoppeld aan een ander object
- Buiten het gegevensmodel werken
- De DataRecalc-handeling gebruiken

*Een variabele gebruiken*

Het volgende voorbeeld toont één manier om een variabele in een berekend veld te gebruiken. De variabele moet binnen het bereik van het berekende veld zijn gedeclareerd. Dat wil zeggen, de variabele moet zijn gedeclareerd in het Var-venster van het veld of van een object dat zich in dat veld bevindt. Ook moet er een waarde aan de variabele zijn toegewezen voordat deze door het veld wordt gebruikt. Het is gebruikelijk om dit te doen in de ingebouwde **open**-methode van het object. In dit voorbeeld is de variabele *mijnWaarde* gedeclareerd in het Var-venster van het veld en is er een waarde aan toegewezen in de **open**-methode van de variabele.

De volgende code is gekoppeld aan het Var-venster van het veld:

```
; Deze code is gekoppeld aan het Var-venster van het veld
Var
   mijnWaarde SmallInt
endVar
```

De volgende code, die aan de ingebouwde **open**-methode van het veld is gekoppeld, wijst een waarde toe aan de variabele *mijnWaarde*:

```
; Deze code is gekoppeld aan de open-methode van het veld
method open(var eventInfo Event)
    mijnWaarde = 12
endmethod
```

De volgende code definieert het berekende veld:

```
mijnWaarde
```

Als dit formulier wordt gestart, wordt de code uitgevoerd die de variabele declareert en een waarde toewijst. De waarde wordt in het berekende veld weergegeven.

*Een eigen methode aanroepen die aan een ander object is gekoppeld*

Het volgende voorbeeld laat zien hoe u een eigen methode aanroept die aan een ander object is gekoppeld en de teruggegeven waarde in een berekend veld toont. U moet een UIObject-variabele binnen het bereik van het berekende veld declareren en een **attach**-instructie gebruiken om de variabele met een object te associëren.

Stel dat in dit voorbeeld een formulier een kader bevat met de naam *hetKader* en dat de volgende eigen methode aan *hetKader* is gekoppeld:

```
; Dit is een eigen methode die aan hetKader is gekoppeld
method eigenMeth() Number ; methode moet een waarde teruggeven
   return 1001.22
endmethod
```

De volgende code, die aan het Var-venster van het berekende veld is gekoppeld, declareert een UIObject-variabele.

```
; Deze code is gekoppeld aan het Var-venster van het veld
Var
   methKader UIObject
endVar
```

De volgende code is gekoppeld aan de ingebouwde **open**-methode van het veld. Hiermee wordt **attach** aangeroepen om de UIObject-variabele *methKader* met het werkelijke UIObject *hetKader* te associëren.

```
; Deze code is gekoppeld aan de open-methode van het veld
method open(var eventInfo Event)
   methKader.attach(hetKader)
endmethod
```

De volgende code definieert het berekende veld:

```
methKader.eigenMeth()
```

Als het formulier wordt gestart, geeft het berekende veld 1001,22 weer, de waarde die door de eigen methode **eigenMeth** wordt teruggegeven.

*Buiten het gegevensmodel werken*

De speciale functies voor berekende velden werken op tabellen in het gegevensmodel van het formulier. In dit voorbeeld wordt getoond hoe u met behulp van een TCursor kunt werken met tabellen buiten het gegevensmodel. U moet een TCursor-variabele declareren en deze openen voordat u de variabele in een berekend veld kunt gebruiken.

De volgende code, die aan het Var-venster is gekoppeld, declareert de TCursor-variabele:

```
; Deze code is gekoppeld aan het Var-venster van het veld
Var
   artikelenTC TCursor
endVar
```

De volgende code is gekoppeld aan de ingebouwde **open**-methode van het veld. Deze code opent een TCursor naar de tabel *Duikspul*.

```
; Deze code is gekoppeld aan de open-methode van het veld
method open(var eventInfo Event)
   artikelenTC.open("duikspul.db")
endmethod
```

De volgende code definieert het berekende veld:

```
artikelenTC.cAverage("Prijs")
```

Als het formulier wordt gestart, geeft het berekende veld het gemiddelde weer van de waarden in het veld 'Prijs' van de tabel *Duikspul*.

*De DataRecalc-handeling gebruiken*

DataRecalc is een handelingsconstante van ObjectPAL. U kunt DataRecalc gebruiken om de handeling waardoor een berekend veld de eigen waarde opnieuw berekent, te initiëren of erop te reageren. Stel dat u een berekend veld hebt, waarin het nummer van het huidige record wordt weergegeven in een tabel buiten het gegevensmodel van het formulier. Stel dat u, terwijl u op het formulier werkt, een TCursor gebruikt om in deze tabel naar waarden te zoeken. Als een zoekactie lukt, wordt het huidige record van de TCursor

veranderd, maar Paradox werkt het berekende veld niet automatisch bij. U moet dat zelf doen.

***Opmerking*** U kunt waarden niet direct, interactief of met behulp van een toewijzingsinstructie van ObjectPAL in berekende velden invoeren. U moet het veld de waarde automatisch opnieuw laten berekenen of een DataRecalc-handeling initiëren. Met andere woorden, als u een berekend veld hebt met de naam *rekenVeld*, mislukt de volgende instructie:

```
rekenVeld.Value = 123 ; deze toewijzing mislukt
```

In het volgende voorbeeld wordt getoond hoe u een DataRecalc-handeling gebruikt om een berekend veld bij te werken. Stel dat in dit voorbeeld een van de objecten op een formulier *artikelRecNum* heet. Dit veldobject toont het nummer van het huidige record in de tabel *Duikspul*, die geen onderdeel vormt van het gegevensmodel van het formulier. Code die aan het Var-venster van het formulier is gekoppeld, declareert de TCursor-variabele *artikelenTC*. Code die aan de ingebouwde **open**-methode van het formulier is gekoppeld, associeert *artikelenTC* met de tabel *Duikspul* en code die aan de ingebouwde **pushButton**-methode van een knop is gekoppeld, initieert een zoekactie. Als de zoekactie lukt, initieert code in de **pushButton**-methode een DataRecalc-handeling die de waarde bijwerkt die in *aantArtikelRecs* wordt weergegeven. De andere code die in dit voorbeeld van belang is, is de code die *aantArtikelRecs* definieert.

De volgende code is gekoppeld aan het Var-venster van het formulier:

```
; Deze code is gekoppeld aan het Var-venster van het formulier
Var
   artikelenTC TCursor
endVar
```

De volgende code is gekoppeld aan de ingebouwde **open**-methode van het formulier:

```
; Deze code is gekoppeld aan de ingebouwde open-methode van het formulier
method open(var eventInfo Event)
   if eventInfo.isPreFilter() then
      ; deze code wordt voor elk object op het formulier uitgevoerd
   else
      ; deze code wordt alleen voor het formulier zelf uitgevoerd
      artikelenTC.open("duikspul.db")
   endIf
endmethod
```

De volgende code definieert het berekende veld:

```
; Deze code definieert het berekende veld
   artikelenTC.recNo()
```

De volgende code is gekoppeld aan de ingebouwde **pushButton**-methode van de knop:

```
; Deze code is gekoppeld aan de ingebouwde pushButton-methode van de knop
method pushButton(var eventInfo Event)
   var
      artikelNr  LongInt
   endVar

   artikelNr = 0
   artikelNr .view("Typ een artikelnummer")

   if artikelNr <> 0 then
      if artikelenTC.locate("ArtikelNr.", artikelNr ) then
         artikelRecNum.action(DataRecalc)
      else
         msgInfo("Zoeken mislukt", String(artikelNr ) + " niet gevonden")
      endIf
   endIf
endmethod
```

Als u dit formulier start, geeft het berekende veld *artikelRecNum* 1 weer. Als u op de knop drukt, wordt u in een **view**-dialoogvenster gevraagd een artikelnummer te typen. Vervolgens wordt dit nummer gezocht in de tabel *Duikspul*. Als het nummer wordt gevonden, werkt de DataRecalc-handeling de waarde bij die in *artikelRecNum* wordt weergegeven.

# Menu: een lijst boven in het venster

Een menu is een lijst met opties die horizontaal over de menubalk van de applicatie verschijnt. De menu's die u met ObjectPAL maakt, vervangen de ingebouwde menu's van Paradox (maar u kunt de Paradox-menu's terugkrijgen met behulp van **removeMenu**). Als u een optie in een menu kiest (een Paradox-menu of een eigen menu), wordt de tekst van die optie teruggegeven aan de ingebouwde **menuAction**-methode. Dit is de methode waarmee u menukeuzes behandelt.

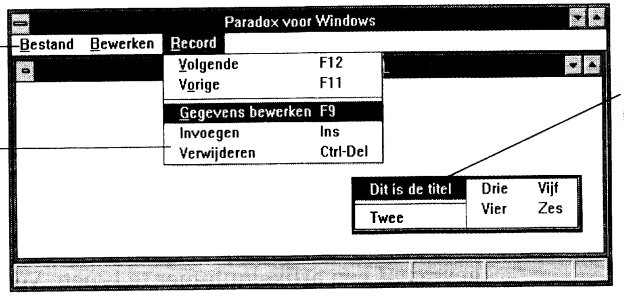
Veel applicaties combineren menu's met pop-up menu's (zoals in Afbeelding 13-12 wordt getoond).

Deze paragraaf begint met een zelfstudie waarin u leert een menu te samen te stellen, weer te geven en erop te reageren. Na de zelfstudie volgen paragrafen waarin geavanceerde werkwijzen worden beschreven voor het werken met menu's en waarin de volgende zaken worden uitgelegd:

- Alleen werken met pop-up menu's
- Werken met de ingebouwde menu's van Paradox en de TurboBalk
- Werken met de systeemmenu's van Windows
- Reageren op opties in de ingebouwde menu's van Paradox

***Belangrijk*** Start de MAST-applicatie en de spelapplicatie Eenarm voor meer voorbeelden van het gebruik van menu's. Beide applicaties bevatten online Helpinformatie en uitleg van de code.

Afbeelding 13-12 Een menu en twee pop-up menu's




## Menu's

Het werken met menu's omvat de volgende basistaken:

- Het menu samenstellen
- Het menu weergeven
- Menukeuzes verwerken

Als u verder wilt gaan dan deze basistaken, kunt u de volgende dingen doen:

- Identificatienummers van menu-opties opgeven
- De weergave-attributen van menu-opties opgeven
- Toetsenbordtoegang verschaffen tot menu-opties
- Opties in een menu inspecteren

In tegenstelling tot de menu's die in Paradox zijn ingebouwd, komen menu's die met ObjectPAL zijn gemaakt, niet terug in alle formulieren in een applicatie. Als u een eigen menu maakt voor een formulier, verschijnt het menu alleen in dat formulier. Als u een ander formulier opent, worden in het tweede formulier standaard de ingebouwde menu's gebruikt en *niet* het menu dat u voor het eerste formulier hebt gemaakt.

***Opmerking*** U kunt twee (of meer) formulieren maken die dezelfde eigen menustructuur gebruiken. In de paragraaf "Geavanceerde menutechnieken", verderop in dit hoofdstuk, worden deze technieken besproken.

In de volgende zelfstudie leert u hoe u het menu uit Afbeelding 13-13 maakt en weergeeft en keuzes in dit menu verwerkt. Deze

basistechnieken worden gepresenteerd in de context van een één-formulier, één-pagina applicatie.

Afbeelding 13-13 Het voltooide menu voor dit deel van de zelfstudie


Deze afbeelding toont de pop-up menu' met de menu-opties zijn geassocieerd. Normaal gesproken kunt u slechts één pop-up menu tegelijk weergeven.

## Menu samenstellen

In dit deel van de zelfstudie leert u code te schrijven om het menu in Afbeelding 13-13 te maken en weer te geven. De eerste stap bestaat uit het maken van een één-record formulier dat is verbonden met de tabel *Klant*.

*Het formulier maken*

1. Kies 'Bestand | Nieuw | Formulier' om het dialoogvenster 'Gegevensmodel' te openen.
2. Voeg KLANT.DB toe aan het gegevensmodel van het formulier. Kies vervolgens 'OK' om het dialoogvenster 'Gegevensmodel' te sluiten. Het dialoogvenster 'Layout ontwerpen' wordt geopend.
3. Klik op 'OK' om de standaard-layout te accepteren en sluit het dialoogvenster.
4. Paradox geeft het nieuwe formulier in een ontwerpvenster weer.

*Code koppelen*

Dit is een één-formulier, één-pagina applicatie. U koppelt de code die het menu maakt, dus aan de pagina. Dit is in het algemeen de beste plaats voor code die een menu maakt. (In een multi-pagina formulier koppelt u de code aan het formulier als u wilt dat alle pagina's gebruik maken van hetzelfde menu.)

1. Inspecteer de pagina en kies 'Methodes' om het methodevenster te openen.
2. Kies **arrive** om een Editor-venster te openen voor de ingebouwde **arrive**-methode van de pagina. De **arrive**-methode is geschikter dan de **open**-methode vanwege de flexibiliteit die deze methode oplevert. Stel dat u later een pagina aan het formulier wilt toevoegen en dat die pagina een ander menu gebruikt. Code die aan **arrive** is gekoppeld, wordt elke keer uitgevoerd als u naar die pagina gaat, terwijl code die aan **open** is gekoppeld, slechts eenmaal wordt uitgevoerd, namelijk als de pagina wordt geopend.
3. Bewerk de methode zodat deze er als volgt uitziet:

```
method arrive(var eventInfo MoveEvent)
   var
      hoofdMenu Menu
```

```
      bestPop, bewerkPop, recordPop PopUpMenu
   endVar

; maak het menu 'Bestand'
   bestPop.addText("Nieuw")
   bestPop.addText("Afsluiten")
   hoofdMenu.addPopUp("Bestand", bestPop)

; maak het menu 'Bewerken'
   bewerkPop.addText("Knippen")
   bewerkPop.addText("Kopiëren")
   bewerkPop.addText("Plakken")
   hoofdMenu.addPopUp("Bewerken", bewerkPop)

; maak het menu 'Record'
   recordPop.addText("Volgende")
   recordPop.addText("Vorige")
   recordPop.addSeparator()
   recordPop.addText("Gegevens bewerken")
   recordPop.addText("Invoegen")
   recordPop.addText("Verwijderen")
   hoofdMenu.addPopUp("Record", recordPop)

; geef het menu weer
   hoofdMenu.show()
endmethod
```

Zoals eerder is gezegd, bestaat een applicatiemenu uit opties die horizontaal op de menubalk worden weergegeven, en de pop-up menu's die met de opties zijn geassocieerd. Deze voorbeeldcode begint met de declaratie van variabelen voor drie pop-up menu's en één variabele voor het hoofdmenu. Vervolgens gebruikt de code **addText**-instructies die opties aan het eerste pop-up menu toevoegen. De opties verschijnen in het menu in de volgorde waarin deze zijn toegevoegd. In dit voorbeeld wordt de optie "Nieuw" het eerst toegevoegd, waardoor deze optie als eerste verschijnt. De optie "Afsluiten" verschijnt eronder.

Nadat alle opties aan een pop-up menu zijn toegevoegd, moet het menu worden geassocieerd met een optie op de menubalk. De volgende instructie roept **addPopUp** aan om het pop-up menu dat wordt vertegenwoordigd door *bestPop*, te associëren met de optie "Bestand" in het hoofdmenu.

```
hoofdMenu.addPopUp("Bestand", bestPop)
```

De rest van het menu wordt op dezelfde manier samengesteld: roep **addText** en (optioneel) **addSeparator** aan om een pop-up menu samen te stellen en voeg het pop-up menu vervolgens toe aan het hoofdmenu.

De aanroep van **addSeparator** voegt een horizontale lijn aan het menu toe om de opties boven en onder de lijn te scheiden.

De aanroep van **show** geeft ten slotte het menu weer .

***Opmerking*** Als u een menu maakt, vervangt dit menu het ingebouwde menu van Paradox. U kunt *geen* opties toevoegen aan de ingebouwde menu's van Paradox.

## Menu weergeven

Zoals u in het vorige voorbeeld hebt gezien, geeft een aanroep van **show** een menu weer. Het nieuwe menu blijft waar het is, totdat u **show** aanroept om een ander menu weer te geven, **removeMenu** aanroept (om de ingebouwde menu's van Paradox te herstellen) of het formulier sluit (hierdoor wordt het ingebouwde menu ook hersteld).

## Menukeuzes verwerken

Als u een optie in het menu kiest, wordt de ingebouwde **menuAction**-methode van het actieve object geactiveerd. Het actieve object laat de actie standaard omhoogborrelen naar het insluitend object. Dit gaat zo door totdat de actie het formulier bereikt en de standaardcode van het formulier de actie behandelt.

U kunt alle code voor de behandeling van menu's koppelen aan het formulier, maar dat gaat ten koste van modulariteit en flexibiliteit. Als u code koppelt aan objecten op een lager niveau (in dit geval aan de pagina), kunt u objecten in en tussen formulieren toevoegen, verwijderen, knippen, kopiëren en plakken zonder dat u grote blokken code op formulierniveau hoeft te onderhouden.

*Code koppelen* De volgende code is gekoppeld aan de ingebouwde **menuAction**-methode van de pagina en behandelt menukeuzes. Als u in het vorige voorbeeld een optie in het menu kiest, geeft Paradox een reeks terug die de gekozen optie bevat. Als u bijvoorbeeld 'Bestand | Nieuw' kiest, geeft Paradox "Nieuw" terug. De *eventInfo*-variabele voor **menuAction** bevat deze informatie en u vraagt deze op met behulp van **menuChoice**.

***Belangrijk*** **menuChoice** geeft de optiereeks terug, op exact dezelfde manier waarop deze in de **addText**-instructie is opgegeven, inclusief hoofdletters, kleine letters, spaties, leestekens en speciale tekens.

```
method menuAction(var eventInfo MenuEvent)
   var
      deKeuze String
      formVar Form
   endVar

   deKeuze = eventInfo.menuChoice()

   switch
   ; menu 'Bestand'
      case deKeuze = "Nieuw"     : formVar.create() ; maak een nieuw formulier
      case deKeuze = "Afsluiten" : close()          ; sluit dit formulier

   ; menu 'Bewerken'
      case deKeuze = "Knippen"   : active.action(EditCutSelection)
      case deKeuze = "Kopiëren"  : active.action(EditCopySelection)
      case deKeuze = "Plakken"   : active.action(EditPaste)
```

```
    ; menu 'Record'
       case deKeuze = "Volgende"          : active.action(DataNextRecord)
       case deKeuze = "Vorige"            : active.action(DataPriorRecord)
       case deKeuze = "Gegevens bewerken" : active.action(DataToggleEdit)
       case deKeuze = "Invoegen"          : active.action(DataInsertRecord)
       case deKeuze = "Verwijderen"       : active.action(DataDeleteRecord)
    endSwitch
endmethod
```

*Werking* De verwerking van menukeuzes verloopt in het algemeen als volgt:

1. Koppel code aan de ingebouwde **menuAction**-methode van de pagina.
2. Roep **menuChoice** aan om te bepalen welke optie is gekozen.
3. Roep de **action**-methode van het UIObject-type aan met een ObjectPAL-constante om een reactie op te geven.

Als u bijvoorbeeld 'Bewerken | Knippen' kiest, activeert de volgende instructie een reactie van het huidige object:

```
active.action(EditCutSelection)
```

De handelingsconstante EditCutSelection zegt in feite "Knip de eventueel geselecteerde tekst uit het actieve object naar het Klembord".

U hoeft **action** en handelingsconstanten niet te gebruiken om op menukeuzes te reageren. U kunt methodes en procedures uit de runtime bibliotheek gebruiken. U kunt methodes aanroepen die aan andere objecten zijn gekoppeld en u kunt eigen methodes en eigen procedures gebruiken. Als u in het bovenstaande voorbeeld bijvoorbeeld 'Bestand | Nieuw' kiest, wordt de volgende instructie uitgevoerd om een nieuw formulier te maken:

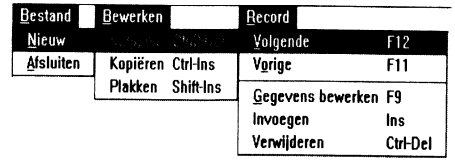
```
formVar.create() ; maak een nieuw formulier
```

## Geavanceerde menutechnieken

In de vorige paragraaf is de basistechniek voor het werken met menu's uitgelegd: het menu samenstellen met **addText** en **addPopUp**, het menu weergeven met **show** en op keuzes reageren met **action**. Deze werkwijze is snel en gemakkelijk, maar heeft wel beperkingen. Twee opties mogen bijvoorbeeld niet dezelfde naam hebben. Stel dat u de menu-opties, 'Bewerken | Kopiëren' en 'Tabel | Kopiëren' wilt toevoegen. Met de basistechniek kunt u deze opties niet van elkaar onderscheiden. Of stel dat u wilt dat een bepaalde menu-optie, afhankelijk van bepaalde gegevens, soms beschikbaar is en soms niet beschikbaar (lichtgekleurd). Stel ten slotte dat u de gebruiker met het toetsenbord toegang wilt verschaffen tot de menu's, dat wil zeggen, gebruikers met het toetsenbord menu-opties laten kiezen. Als u deze taken wilt uitvoeren, moet u de technieken gebruiken die in deze paragraaf worden besproken.

In deze paragraaf maakt u het volledig functionerende menu uit Afbeelding 13-14. Dit menu is vrijwel identiek aan het menu in het vorige voorbeeld, maar u gebruikt de geavanceerde functies om de functionaliteit te vergroten.

Afbeelding 13-14 Een volledig menu




Aan het eind van deze paragraaf wordt aan de hand van een voorbeeld getoond hoe u reageert op keuzes in de ingebouwde Paradox-menu's.

### *Identificatienummers toewijzen aan menu-opties*

De eerste stap bij het maken van een menu voor een complexe applicatie is de toekenning van identificatienummers aan de menu-opties. U kunt dan gehele getallen (in plaats van tekenreeksen) gebruiken om menukeuzes te onderscheiden. Als u identificatienummers toewijst, kunt u code schrijven die wordt uitgevoerd ongeacht de tekst in de menu-opties: u kunt dubbele opties gebruiken, u kunt opties wijzigen en u kunt de opties zelfs in een andere taal vertalen zonder dat de onderliggende code wordt beïnvloed.

Begin met de definitie van enkele constanten. Koppel de volgende code aan het Const-venster van de pagina.

```
Const
   BestandNieuw = 101
   BestandAfsluiten = 102

   BewerkenKnippen = 201
   BewerkenKopiëren = 202
   BewerkenPlakken = 203

   RecordVolgende = 301
   RecordVorige = 302
   RecordBewerken = 303
   RecordInvoegen = 304
   RecordVerwijderen = 305
endConst
```

Deze code kent gehele getallen toe aan constanten die menukeuzes vertegenwoordigen. De constante BestandNieuw vertegenwoordigt bijvoorbeeld de menukeuze 'Bestand | Nieuw'. De vaste getalwaarden hebben alleen betekenis als geheugensteun. De waarde 101 vertegenwoordigt de eerste optie in het eerste menu; 201 is de eerste optie in het tweede menu enzovoort.

### addText optimaal benutten

Nu u de constanten hebt gedefinieerd, kunt u deze in **addText**-instructies gebruiken om de pop-up menu's en de menu's te maken. De uitgebreide syntaxis van **addText** gebruikt drie argumenten: de tekst van de menu-optie (zoals eerder), een identificatienummer en een argument dat het weergave-attribuut van de optie aangeeft. Als u een optie bijvoorbeeld lichtgekleurd of geselecteerd wilt weergeven, kunt u dat doen.

U kunt **addText** ook gebruiken om de gebruiker op een van de volgende manieren toegang te geven met het toetsenbord:

- Combinaties van *Alt* en andere toetsen. Als u een ampersand (&) voor een letter in een menu-optie plaatst, geeft u aan op welke toets moet worden gedrukt in combinatie met *Alt*.
- Toegangstoetsen. Een *toegangstoets* is een functietoets of toetscombinatie waarop u drukt om toegang te krijgen tot een menu-optie. Het maken van een toegangstoets vereist twee stappen: eerst plaatst u de toegangstoets met **addText** in de menu-optie. Daarna schrijft u code om de toetsaanslag in een handeling te vertalen (dit wordt verderop beschreven).

In het volgende voorbeeld wordt code getoond waarmee u dit allemaal kunt doen. Daarna volgt een uitleg.

```
method arrive(var eventInfo MoveEvent)
   var
      hoofdMenu Menu
      bestPop, bewerkPop, recordPop PopUpMenu
   endVar

; maak het menu 'Bestand'
   bestPop.addText("&Nieuw", MenuEnabled, UserMenu + BestandNieuw)
   bestPop.addText("&Afsluiten", MenuEnabled, UserMenu + BestandAfsluiten)
   hoofdMenu.addPopUp("&Bestand", bestPop)

; maak het menu 'Bewerken'
   bewerkPop.addText("Knippen\tShift-Del", MenuGrayed + MenuDisabled,
                                                UserMenu + BewerkenKnippen)
   bewerkPop.addText("Kopiëren\tCtrl-Ins", MenuGrayed + MenuDisabled,
                                                UserMenu + BewerkenKopiëren)
   bewerkPop.addText("Plakken\tShift-Ins", MenuGrayed + MenuDisabled,
                                                UserMenu + BewerkenPlakken)
   hoofdMenu.addPopUp("&Bewerken", bewerkPop)

; maak het menu 'Record'
   recordPop.addText("&Volgende\tF12", MenuEnabled, UserMenu + RecordVolgende)
   recordPop.addText("V&orige\tF11", MenuEnabled, UserMenu + RecordVorige)
   recordPop.addSeparator()
   recordPop.addText("&Gegevens bewerken\tF9", MenuEnabled, UserMenu +
RecordBewerken)
   recordPop.addText("Invoegen\tIns", MenuGrayed + MenuDisabled,
                                                UserMenu + RecordInvoegen)
   recordPop.addText("Verwijderen\tCtrl-Del", MenuGrayed + MenuDisabled,
                                                UserMenu + RecordVerwijderen)
   hoofdMenu.addPopUp("&Record", recordPop)

; geef het menu weer
```

```
   hoofdMenu.show()
endmethod
```

Laat u niet afschrikken door de hoeveelheid code die hier wordt gebruikt. Als u de volgende regel begrijpt, begrijpt u alle regels:

```
recordPop.addText("&Gegevens bewerken\tF9", MenuEnabled, UserMenu +
RecordBewerken)
```

Het eerste gedeelte, **recordPop.addText**, hebt u eerder gezien. Hiermee wordt **addText** aangeroepen om een optie toe te voegen aan het pop-up menu dat wordt vertegenwoordigd door de variabele *recordPop*.

Het volgende gedeelte, **&Gegevens bewerken\tF9**, wijst de reeks "Gegevens bewerken" aan de optie toe. De ampersand voor de letter G betekent dat u op *Alt-G* kunt drukken om toegang te krijgen tot deze optie (deze mogelijkheid is in Windows ingebouwd; u hoeft dit dus niet zelf te programmeren). De tekens **\tF9** kennen *F9* als toegangstoets toe. Misschien herkent u "\t": het is de backslash-code voor een tab-teken. U moet code schrijven om deze toegangstoets te activeren, zoals later wordt uitgelegd.

Het volgende gedeelte, **MenuEnabled**, is een ObjectPAL-constante. Deze constante zorgt ervoor dat de optie normaal verschijnt en dat de gebruiker deze kan kiezen. U kunt meer constanten toevoegen. Met MenuGrayed + MenuDisabled maakt u de optie bijvoorbeeld lichtgekleurd, waardoor deze niet kan worden gekozen. In Tabel 13-4 wordt een overzicht gegeven van de ObjectPAL-constanten die attributen van menukeuzes instellen.

Het laatste gedeelte, **UserMenu + RecordBewerken,** geeft een constante (die eerder is gedefinieerd) op om deze menu-optie te identificeren. Deze constante wordt in de ingebouwde **menuAction**-methode (zie verderop) gebruikt.

U kunt uw eigen constanten definiëren om menu-opties te identificeren, maar er is wel een beperking: u moet de constanten binnen een bepaald bereik houden. Omdat dit bereik in toekomstige versies van Paradox kan veranderen, kent ObjectPAL de constanten UserMenu en MaxUserMenu. Deze constanten vertegenwoordigen de toegestane minimale en maximale waarden. Als u UserMenu bij uw eigen constante optelt, bent u zeker van een waarde boven het minimum. Als u de waarde onder het maximum wilt houden, moet u de waarde kennen van MaxUserMenu. Eén manier om deze waarde te bepalen is een **message**-instructie te gebruiken, zoals hieronder.

```
message(MaxUserMenu).
```

In deze versie van Paradox is het verschil tussen UserMenu en MaxUserMenu 2047. Dat betekent dat de grootste waarde die u voor het identificatienummer van een menu kunt gebruiken, UserMenu +

2047 is. Definieer dus geen menuconstante met een waarde groter dan 2047.

Zoals u ziet, kunt u met de uitgebreide syntaxis van **addText** veel aspecten van de weergave en de mogelijkheden van een menu instellen.

Tabel 13-4 Menukeuze-attributen

| Constante | Beschrijving |
|---|---|
| MenuChecked | Geeft de optie weer, voorafgegaan door een vinkje |
| MenuDisabled | Maakt de optie niet-actief |
| MenuEnabled | Maakt de optie actief |
| MenuGrayed | Geeft de optie in grijze (lichtgekleurde) tekens weer |
| MenuHilited | Geeft de optie gemarkeerd weer |
| MenuNotChecked | Geeft de optie zonder vinkje weer |
| MenuNotGrayed | Geeft de optie normaal weer |
| MenuNotHilited | Geeft de optie zonder markering weer |

### Menukeuzes verwerken door middel van identificatienummers

Eerder in dit hoofdstuk hebt u in een voorbeeld gezien dat Paradox de tekst teruggeeft van een optie die in een menu is gekozen. Paradox geeft ook een geheel getal terug dat het identificatienummer van de gekozen menu-optie vertegenwoordigt. In deze paragraaf wordt uitgelegd hoe u met identificatienummers van menu's werkt.

De volgende code, die aan de ingebouwde **menuAction**-methode van de pagina is gekoppeld, verwerkt menukeuzes met behulp van identificatienummers. Deze code gebruikt **id**, gedefinieerd voor het MenuEvent-type, om het identificatienummer van de gekozen menu-optie te bepalen, die is opgeslagen in *eventInfo*. Vervolgens wordt een groot **switch**-blok gebruikt om een juiste reactie op elke menukeuze op te geven.

```
method menuAction(var eventInfo MenuEvent)
  var
     optieID SmallInt
     formVar Form
  endVar

  optieID = eventInfo.id()

  switch
  ; menu 'Bestand'
     case optieID = UserMenu + BestandNieuw  : formVar.create()
     case optieID = UserMenu + BestandAfsluiten : close()

  ; menu 'Bewerken'
     case optieID = UserMenu + BewerkenKnippen  : active.action
                                                  (EditCutSelection)
     case optieID = UserMenu + BewerkenKopiëren : active.action
                                                  (EditCopySelection)
     case optieID = UserMenu + BewerkenPlakken  : active.action(EditPaste)
```

```
    ; menu 'Record'
       case optieID = UserMenu + RecordVolgende  : active.action(DataNextRecord)
       case optieID = UserMenu + RecordVorige    : active.action(DataPriorRecord)
       case optieID = UserMenu + RecordBewerken  : active.action(DataToggleEdit)
       case optieID = UserMenu + RecordInvoegen  : active.action
                                                          (DataInsertRecord)
       case optieID = UserMenu + RecordVerwijderen : active.action
                                                          (DataDeleteRecord)
    endSwitch
endmethod
```

De verwerking van menukeuzes via identificatienummers lijkt veel op de verwerking met behulp van optiereeksen. U moet wel meer code schrijven als u met identificatienummers wilt werken, maar uw code wordt, ongeacht de tekst van de menu-opties, uitgevoerd.

### Attributen van menu-opties instellen

De code in de paragraaf "addText optimaal benutten" liet zien hoe u ObjectPAL-constanten met **addText** gebruikt om te bepalen hoe een optie wordt weergegeven. U kunt deze constanten ook gebruiken als het menu wordt weergegeven. Meestal zult u de weergave van een optie willen laten afhangen van voorwaarden op het formulier. Als de gebruiker bijvoorbeeld op *F9* drukt of in uw eigen menu 'Record | Gegevens bewerken' kiest om de bewerkmodus te activeren of te verlaten, wilt u de menu-optie 'Gegevens bewerken' bijvoorbeeld selecteren of deselecteren om de huidige toestand van het formulier weer te geven. Stel dat u bovendien de opties 'Record | Invoegen' en 'Record | Verwijderen' wilt activeren als de bewerkmodus actief is en dat u deze opties wilt uitschakelen als de bewerkmodus niet actief is. De volgende code, die aan de ingebouwde **action**-methode van de pagina is gekoppeld, laat zien hoe u dit doet.

```
method action(var eventInfo ActionEvent)
   var
      RecordBewerkAttrib LongInt
   endVar
   if eventInfo.id() = DataToggleEdit then
      RecordBewerkAttrib = getMenuChoiceAttributeByID(RecordBewerken)
      if hasMenuChoiceAttribute(RecordBewerkAttrib, MenuChecked) then
         setMenuChoiceAttributeByID(UserMenu + RecordBewerken, MenuNotChecked)
         setMenuChoiceAttributeByID(UserMenu + RecordInvoegen,
                                                MenuGrayed + MenuDisabled)
         setMenuChoiceAttributeByID(UserMenu + RecordVerwijderen,
                                                MenuGrayed + MenuDisabled)
      else
         setMenuChoiceAttributeByID(UserMenu + RecordBewerken, MenuChecked)
         setMenuChoiceAttributeByID(UserMenu + RecordInvoegen, MenuEnabled)
         setMenuChoiceAttributeByID(UserMenu + RecordVerwijderen, MenuEnabled)
      endIf
   endIf
endmethod
```

U kunt deze werkwijze ook gebruiken om de menukeuze-attributen van andere objecten in te stellen. U kunt bijvoorbeeld code koppelen aan de ingebouwde **changeValue**-methode van een veldobject om een menukeuze beschikbaar of niet beschikbaar te maken, afhankelijk van de waarde van het veldobject.

*Menukeuze-attributen instellen met MenuInit*

Als u een optie kiest op de menubalk, wordt er, voordat het geassocieerde pop-up menu wordt weergegeven, een handeling geïnitieerd die wordt vertegenwoordigd door de constante MenuInit. Als u bijvoorbeeld 'Bestand | Nieuw' kiest in uw eigen menu, is de actievolgorde als volgt:

1. U klikt op het woord "Bestand" op de menubalk.
2. Paradox initieert een MenuInit-menuhandeling en een MenuEvent die de ingebouwde **menuAction**-methodes van objecten op het formulier activeert.
3. Paradox opent het pop-up menu dat is geassocieerd met de menu-optie 'Bestand'.
4. U kiest "Nieuw" in het pop-up menu, waardoor een MenuEvent wordt gegenereerd.
5. De MenuEvent activeert weer de ingebouwde **menuAction**-methodes.

Met andere woorden, MenuInit vertegenwoordigt het moment voordat het pop-up menu verschijnt, het moment waarop u attributen van menu-opties kunt initialiseren voordat deze aan de gebruiker worden getoond. In het eigen menu dat u in het eerste gedeelte van deze paragraaf hebt gemaakt, worden de menu-opties 'Bewerken | Knippen' en 'Bewerken | Kopiëren' bijvoorbeeld lichtgekleurd en niet beschikbaar gemaakt in de oorspronkelijke **addText**-instructie. Als u de volgende code aan de ingebouwde **menuAction**-methode van de pagina toevoegt, maakt u deze opties beschikbaar als de gebruiker tekst in een veldobject selecteert. In dit voorbeeld gaat de code die MenuInit behandelt, vooraf aan het **switch**-blok dat de eigenlijke menukeuzes behandelt. Deze volgorde wordt aanbevolen omwille van de leesbaarheid en omdat zo de opeenvolging van acties in Paradox duidelijk wordt.

```
method menuAction(var eventInfo MenuEvent)

   var
      optieID SmallInt
      terugWaarde Logical
   endVar

optieID = eventInfo.id()

if optieID = MenuInit then
   try
      terugWaarde = active.Editing ; 'Editing'-kenmerk moet True zijn
   onFail                          ; om toegang te krijgen tot het kenmerk
                                   ; 'SelectedText'
      return
   endTry

   if terugWaarde = True then
      if active.SelectedText <> "" then
         setMenuChoiceAttributeByID(UserMenu + BewerkenKnippen, MenuEnabled)
```

```
            setMenuChoiceAttributeByID(UserMenu + BewerkenKopiëren, MenuEnabled)
         else
            setMenuChoiceAttributeByID(UserMenu + BewerkenKnippen,
                                       MenuGrayed + MenuDisabled)
            setMenuChoiceAttributeByID(UserMenu + BewerkenKopiëren,
                                       MenuGrayed + MenuDisabled)
         endIf
      endIf
   endIf
   { het switch-blok dat de menukeuzes behandelt, komt hier }

   endmethod
```

U kunt MenuInit gebruiken om te testen op de kenmerken van een object op het formulier en om de menukeuze-attributen aan de hand van deze kenmerken in te stellen.

## Toegangstoetsen

Een toegangstoets is een functietoets of toetscombinatie waarop de gebruiker kan drukken om toegang te krijgen tot een bepaalde optie in een bepaald menu. De toegangstoets *Ctrl-Ins* heeft bijvoorbeeld hetzelfde effect als het kiezen van 'Bewerken | Kopiëren' (maar het menu wordt niet afgerold). Het maken van een toegangstoets vereist twee stappen:

1. Voeg de toegangstoets aan de menu-optie toe.
2. Schrijf een methode om de toetsaanslag in een menukeuze te vertalen.

In de volgende paragrafen worden deze stappen gedemonstreerd aan de hand van een eenvoudig menu en niet in de context van het complexe menu dat in eerdere voorbeelden werd gebruikt.

### *Stap 1: een toegangstoets aan een menu-optie toevoegen*

De eerste stap is "\t" toevoegen om een tab te plaatsen tussen een optie en de bijbehorende toegangstoets. Bijvoorbeeld:

```
recordPop.addText("Volgende\tF12")
recordPop.addText("Vorige\tF11")
```

verschijnt als

```
Volgende     F12
Vorige       F11
```

Op *F12* drukken is hetzelfde als 'Openen' kiezen en op *F11* drukken is hetzelfde als 'Nieuw' kiezen. Net als ampersands maakt "\t" deel uit van de teruggegeven waarde. Als u wilt bepalen welke keuze de gebruiker heeft gemaakt, gaat u dus bijvoorbeeld als volgt te werk:

```
method menuAction(var eventInfo MenuEvent)
   var
      optieID String
   endVar

   optieID = eventInfo.menuChoice()
   switch
      case optieID = "Volgende\tF12" : active.action(DataNextRecord)
```

```
      case optieID = "Vorige\tF11"   : active.action(DataPriorRecord)
   endSwitch
endMethod
```

*Stap 2: de toetsaanslag in een menukeuze vertalen*

De tweede stap is het schrijven van een methode die de juiste handeling uitvoert als de gebruiker op een toegangstoets drukt. Pas hiervoor de ingebouwde **keyPhysical**-methode van het formulier aan.

***Opmerking***

Dit voorbeeld legt alleen uit *hoe* u te werk dient te gaan. Als u wilt weten *waarom,* kunt u Hoofdstuk 12 en Appendix B raadplegen.

1. Klik rechts op de titelbalk van het formulier om het te inspecteren en open een venster van de ObjectPAL-Editor om de **keyPhysical**-methode te bewerken.
2. Bewerk **keyPhysical** zodat deze methode er als volgt uitziet:

```
method keyPhysical(var eventInfo KeyEvent)
   var
      deToets String
   endvar

if eventInfo.isPreFilter() then
   disableDefault
   deToets = eventInfo.vChar()
   switch
      case deToets = "VK_F12" : active.action(DataNextRecord)
      case deToets = "VK_F11" : active.action(DataPriorRecord)
      otherwise               : doDefault
   endswitch
endif
endmethod
```

In dit voorbeeld worden virtuele toetscodes gebruikt (beschreven in de paragraaf "KeyEvent" in Hoofdstuk 12). ObjectPAL kent constanten voor virtuele Windows-toetsen (zoals *F11* en *F12*). De online ObjectPAL Help geeft een overzicht. Als u de lijst wilt bekijken, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten'. Kies vervolgens in de kolom 'Types constanten' de optie 'Keyboard'. De constanten verschijnen in de kolom 'Constanten'.

## Reageren op keuzes in ingebouwde Paradox-menu's

In de vorige paragrafen is besproken hoe u keuzes in eigen menu's maakt en reageert op keuzes in deze menu's. In deze paragraaf wordt uitgelegd hoe u reageert op keuzes in de ingebouwde Paradox-menu's. Met de werkwijze die hier wordt uitgelegd, kunt u voorkomen dat u code voor een volledig menusysteem moet schrijven terwijl u slechts enkele speciale functies nodig hebt. Stel dat u een eigen dialoogvenster wilt weergeven als de gebruiker 'Bestand | Afsluiten' kiest. U kunt geen eigen opties aan de ingebouwde menu's toevoegen, dus u zou voor deze eigenschap een heel menusysteem moeten maken. U kunt echter ook de onderstaande aanpak gebruiken om te reageren als de gebruiker 'Bestand | Afsluiten' kiest in het ingebouwde menu. De code in dit voorbeeld is gekoppeld aan de ingebouwde **menuAction**-methode van de pagina:

```
method menuAction(var eventInfo MenuEvent)
   var
      eigenDlg Form
      dlgKeuze AnyType
   endVar

   if eventInfo.id() = MenuFileExit then
      eigenDlg.open("dlg.fdl")
      dlgKeuze = eigenDlg.wait()
      doDefault
   endIf
endmethod
```

Deze code gebruikt de **id**-methode van het MenuEvent-type en de MenuCommand-constante MenuFileExit om op de waarde van de gekozen menu-optie te testen. De MenuCommand-constanten zijn gedefinieerd als de waarden die worden teruggegeven door de overeenkomende opties in een ingebouwd menu. MenuFileExit vertegenwoordigt de menu-optie 'Bestand | Afsluiten'. Andere voorbeelden zijn de constante MenuEditCopy, die 'Bewerken | Kopiëren' vertegenwoordigt en de constante MenuHelpAbout, die 'Help | Info' vertegenwoordigt.

Een ander voorbeeld van het gebruik van MenuCommand-constanten: stel dat u een formulier hebt dat vol staat met objecten en dat de ingebouwde menu's voor alle objecten precies doen wat u wilt, behalve voor één object, een memoveld met de naam *memoVeld*. Voor dit memoveld wilt u dat 'Bewerken | Kopiëren' de inhoud van het veld naar een bestand schrijft in plaats van naar het Klembord. De volgende code, die aan de ingebouwde **menuAction**-methode van de pagina is gekoppeld, toont een manier om dit te doen.

```
method menuAction(Var eventInfo MenuEvent)
   var
      hetMemo Memo
   endVar

   if eventInfo.id() = BewerkenKopiëren then
      hetMemo = memoVeld.value
      hetMemo.writeToFile("memoData.txt")
   endIf
endmethod
```

Gebruik MenuCommand-constanten om te testen op menuhandelingen en erop te reageren. Als u een menuhandeling wilt initiëren, gebruikt u de **action**-methode of andere methodes of procedures. Het System-type bevat verschillende procedures die ingebouwde dialoogvensters van Paradox weergeven voor taken als het toevoegen en kopiëren van tabellen. Deze procedures beginnen met de letters **dlg**, bijvoorbeeld **dlgAdd** en **dlgCopy**. Raadpleeg de online ObjectPAL Help voor meer informatie en voorbeelden.

# PopUpMenu: lijsten op verzoek

Een PopUpMenu is een verticale lijst met opties die verschijnt in reactie op een actie (gewoonlijk een klik met de muis). Als de gebruiker een optie in een pop-up menu kiest, wordt de tekst van die optie aan de methode teruggegeven.

Een PopUpMenu verschilt van een Menu, een horizontale lijst met opties die op de menubalk van de applicatie verschijnt.

***Opmerking*** Als u een optie in een pop-up menu kiest, wordt de ingebouwde **menuAction**-methode *niet* geactiveerd.

U kunt PopUpMenu-methodes gebruiken voor de volgende taken:

- Een pop-up menu samenstellen
- Het pop-up menu weergeven en de geselecteerde optie teruggeven
- De opties in een pop-up menu inspecteren
- Toetsenbordtoegang verschaffen

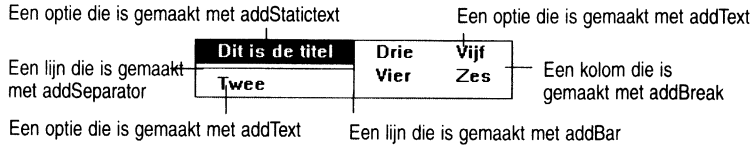
## Pop-up menu samenstellen

Gebruik **addArray**, **addText**, **addStaticText**, **addSeparator**, **addBar** en **addBreak** om een pop-up menu samen te stellen. Gebruik **show** om het pop-up menu weer te geven en de geselecteerde optie terug te geven.

In het volgende voorbeeld wordt getoond hoe u het pop-up menu uit Afbeelding 13-15 maakt en weergeeft. Waarschijnlijk zult u nooit een dergelijk menu willen maken. De bedoeling van deze afbeelding is echter om in een oogopslag de werking te laten zien van de verschillende manieren om menu's te maken.

```
var
   p PopUpMenu
   deKeuze String
endVar

p.addStaticText("Dit is de titel")
p.addSeparator()
p.addText("Twee")
p.addBar()
p.addText("Drie")
p.addText("Vier")
p.addBreak()
p.addText("Vijf")
p.addText("Zes")

deKeuze = p.show()
; Geeft het menu weer, plaatst de keuze van de gebruiker in deKeuze.
; Volgende opdrachten en methodes worden uitgevoerd op basis van de
; waarde van deKeuze.
```

Afbeelding 13-15 Voorbeeld van een pop-up menu




### *addArray*

Met **addArray** kunt u hetzelfde effect bereiken als met meerdere **addText**-instructies. De code in voorbeeld 1 produceert bijvoorbeeld hetzelfde pop-up menu als de code in voorbeeld 2:

*Voorbeeld 1*

```
var
   ar Array[3] String
   p PopUpMenu
   x String
endVar
ar[1] = "een"
ar[2] = "twee"
ar[3] = "drie"
p.addArry(ar)
x = p.show()
```

*Voorbeeld 2*

```
var
   p PopUpMenu
   x String
endVar
p.addText("een")
p.addText("twee")
p.addText("drie")
x = p.show()
```

Er is niet veel verschil tussen deze twee voorbeelden, omdat u de array en de toegevoegde opties met de methode maakt. Als u **addArray** volledig wilt benutten, maakt u de array elders. Gebruik bijvoorbeeld een **copyToArray**-instructie om de velden van een tabel naar een array te kopiëren. Daarna kunt u deze in een pop-up menu weergeven.

### *addPopUp*

**addPopup** voegt één pop-up menu aan een ander pop-up menu toe, zodat een vervolgmenu ontstaat. In het volgende voorbeeld is *p2* een pop-up menu dat drie opties bevat: Eerste, Tweede en Derde. Als u Derde kiest, verschijnt een tweede pop-up menu (*p1*).

```
var
   p1, p2 PopUpMenu
   x String
endVar
p1.addText("een")
p1.addText("twee")

p2.addText("Eerste")
p2.addText("Tweede")
p2.addPopUp("Derde", p1)

x = p2.show()
```

### *Toetsenbordtoegang*

De manier om toetsenbordtoegang te verschaffen tot een pop-up menu is hetzelfde als bij een menu. Zie de paragraaf "Menu: een lijst boven in het venster", in dit hoofdstuk, voor meer informatie.

### *Opties inspecteren in een pop-up menu*

Gebruik **contains** en **count** om de opties in een pop-up menu te inspecteren. Deze methodes zijn handig als u gaandeweg een pop-up menu maakt. Gebruik **remove** om opties te verwijderen.

### switchMenu: een snelle manier

Het PopUpMenu-type kent de **switchMenu**-structuur, een snelle manier om pop-up menu's te maken en weer te geven. Deze structuur gebruikt de functies van **addText** en **show** met een **switch...endSwitch**-blok. De case-instructies links van de dubbele punt geven de menu-opties op die moeten worden weergegeven en de instructies rechts van de dubbele punt worden afhankelijk van de gekozen optie uitgevoerd.

Het volgende voorbeeld geeft een pop-up menu met twee opties weer, *optieEen* en *optieTwee*. U kunt beide opties met de muis kiezen en u kunt *optieEen* met het toetsenbord kiezen door op *Alt-E* te drukken. Als u een optie kiest, verschijnt een dialoogvenster; anders klinkt er een pieptoon.

```
switchMenu
   case "optie&Een" : msgInfo("U koos:", "optieEen")
   case "optieTwee"  : msgInfo("U koos:", "optieTwee")
   otherwise        : beep()
endSwitchMenu
```

## Ingebouwde menu's en de TurboBalk

U kunt **menuAction** en **id** gebruiken om met ingebouwde menu's van Paradox te werken. Stel dat u een bericht wilt weergeven als de gebruiker uw applicatie sluit. Als u de ingebouwde Paradox-menu's gebruikt, kunt u als volgt te werk gaan om de **menuAction**-methode van een object te wijzigen:

```
method menuAction(var eventInfo MenuEvent)
   if eventInfo.id() = MenuFileExit then
      msgInfo("Dag", "Dank u.")
   endIf
endMethod
```

ObjectPAL kent MenuCommand-constanten (zoals MenuFileExit) die de numerieke waarden van de ingebouwde menu-opties vertegenwoordigen. De constanten kunnen online worden weergegeven. Als u de lijst wilt bekijken, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten'. Vervolgens kiest u 'MenuCommands' in de kolom 'Types constanten'. De constanten verschijnen in de kolom 'Constanten'.

In plaats van constanten kunt u ook **menuChoice** gebruiken om te testen op de tekstreeks die voor elke menukeuze wordt teruggegeven. Als u bijvoorbeeld 'Bewerken | Kopiëren' kiest, is de teruggegeven reeks "Kopiëren".

```
method menuAction(var eventInfo MenuEvent)
if eventInfo.menuChoice() = "Kopiëren" then
  msgInfo("Kopiëren", "U koos Kopiëren.")
else
   msgInfo("U koos:", eventInfo.menuChoice())
endIf
endMethod
```

***Opmerking*** Als twee menu-opties dezelfde reeks teruggeven ('Bewerken | Kopiëren' en 'Bestand | Hulpmiddelen | Kopiëren' geven bijvoorbeeld beide "Kopiëren" terug), werkt **menuChoice** met de eerste, waarbij van boven naar beneden en van links naar rechts wordt geteld.

## TurboBalk

De menutechnieken die in dit hoofdstuk zijn besproken, werken ook met TurboBalk-knoppen die menukeuzes vertegenwoordigen, omdat de knoppen sneltoetsen zijn voor hetzelfde eindresultaat. Als u op de TurboBalk bijvoorbeeld op de knop 'Knippen naar klembord' klikt, is dat hetzelfde als wanneer u 'Bewerken | Knippen' kiest. U hoeft dus geen aparte methodes te schrijven voor de behandeling van TurboBalk-knoppen. U kunt dezelfde **menuAction**-methode gebruiken om beide acties te behandelen.

In het volgende voorbeeld geeft **eventInfo.id** MenuEditCut terug, ongeacht of u de menu-optie kiest of op de knop op de TurboBalk klikt.

```
method menuAction(var eventInfo MenuEvent)
if eventInfo.id() = MenuEditCut then
   doeBewerkenKnippen() ; voer een eigen methode uit
endIf
endMethod
```

***Opmerking*** In het standaardgedrag van de **menuAction**-methode van een formulier wordt de menuhandeling geconverteerd naar een handeling en wordt de **action**-methode geactiveerd met de overeenkomende ID-handeling. Stel dat het formulier een menuhandeling behandelt en dat de identificatie MenuEditCut is. Het formulier converteert deze menuhandeling standaard naar een handeling met de identificatie EditCutSelection. Met andere woorden, als u een optie kiest in een menu, activeert u de **menuAction**-methode die de **action**-methode activeert.

## Systeemmenu's

U kunt **id-** en MenuCommand-constanten in de ingebouwde **menuAction**-methode van een object gebruiken om te testen op keuzes (zoals 'Pictogram' en 'Maximumvenster') in het Systeemmenu van Windows. De volgende code voorkomt dat u het huidige formulier tot een pictogram verkleint.

```
method menuAction(var eventInfo MenuEvent)
   if eventInfo.id() = MenuSystemMinimize then
                     ; MenuSystemMinimize is een constante
      disableDefault ; voer de standaardcode niet uit
      beep()
      message("Kan dit formulier niet tot pictogram verkleinen.")
   endIf
endMethod
```

## Geavanceerd voorbeeld van een menu

Dit voorbeeld neemt menuhandelingen op en laat de gebruiker op een knop drukken om de laatste menu-opdracht af te spelen. Neem aan dat dit formulier een veld heeft (bijvoorbeeld een memoveld), een knop en een kader dat een tekstvak insluit. De volgende code is gekoppeld aan het Var-venster van het formulier. De variabelen die in dit Var-venster zijn gedeclareerd, zijn globaal voor elk object op het formulier.

```
Var
  laatsteMenuId    SmallInt   ; ID van het laatst gekozen menu
  laatsteDoel      UIObject   ; handle voor het laatste doel
endVar
```

De volgende code is gekoppeld aan de **open**-methode van het formulier. Deze code maakt een eenvoudig menu, waarmee de gebruiker geselecteerde tekst in het veld kan knippen, plakken of verwijderen.

```
; ditFormulier::open
method open(var eventInfo Event)
var
  menu1    Menu
  popUp1   PopUpMenu
endVar
if eventInfo.isPreFilter()
  then
    ;deze code wordt voor elk object op het formulier uitgevoerd
  else
    ;deze code wordt alleen voor het formulier zelf uitgevoerd
    popUp1.addText("&Knippen")             ; maakt een pop-up menu
    popUp1.addText("&Plakken")
    popUp1.addText("&Verwijderen")
    menu1.addPopUp("&Bewerken", popUp1)    ; koppel pop-up aan een optie op de
menubalk
    menu1.show()                           ; toon het menu
endif
endMethod
```

De volgende code is gekoppeld aan de ingebouwde **menuAction**-methode van het formulier. Deze code controleert het laatst geselecteerde menu en het object dat focus had op het moment dat de MenuEvent plaatsvond. Deze informatie wordt opgeslagen zodat de ingebouwde **pushButton**-methode van de knop weet welke menu-opdracht moet worden uitgevoerd en op welk object de menu-opdracht moet worden uitgevoerd.

```
; ditFormulier::menuAction
method menuAction(var eventInfo MenuEvent)
if eventInfo.isPreFilter()
```

```
  then
    ;deze code wordt voor elk object op het formulier uitgevoerd
    if eventInfo.isFromUI() then          ; als gebruiker een menu kiest
      laatsteMenuId = eventInfo.id()      ; sla de ID van het geselecteerde
menu op
      eventInfo.getTarget(laatsteDoel)    ; sla de huidige doel-handle op naar
                                          ; de variabele laatsteDoel
    endif
  else
    ;deze code wordt alleen voor het formulier zelf uitgevoerd
endif
endmethod
```

De volgende code is gekoppeld aan de ingebouwde **pushButton**-methode van de knop. Als de gebruiker op deze knop drukt, maakt de code een actie van de laatste MenuEvent en roept de code vervolgens de ingebouwde **menuAction**-methode aan die bij het laatste doel hoort. Deze code speelt eigenlijk de meest recente menu-opdracht af.

```
; afspeelKnop::pushButton
method pushButton(var eventInfo Event)
var
  laatsteMenuActie MenuEvent              ; maakt een MenuEvent
endVar

 ; als gebruiker op de knop drukt voordat een menu is gekozen,
 ; vertel de gebruiker dan wat er aan de hand is
if NOT isAssigned(laatsteMenuId) then
  msgStop("Stop", "U moet een menu kiezen voordat u het opnieuw kunt afspelen.")
else                ; gebruiker moet ten minste eenmaal een menu hebben gekozen
  laatsteMenuActie.setId(lastMenuId)
                    ; stel de id van de menu-actie in op de laatste menu-id
  laatsteDoel.menuAction(laatsteMenuActie)
                    ; roep menuAction-methode van het laatste doel aan
endif
endmethod
```

De volgende code is gekoppeld aan de ingebouwde **menuAction**-methode van het veld. Dit is de plaats waar menuselecties worden bekeken en uitgevoerd. Bij wijze van demonstratie verandert deze code ook de tekst en de kleur van het kader om aan te geven waar de MenuEvent vandaan komt.

```
; veldEen::menuAction
method menuAction(var eventInfo MenuEvent)
var
  keuze String
endVar

 ; verander de kleur van het kader en de tekst in het kader om
 ; aan te geven waar de MenuEvent vandaan komt
kaderEenTekst.value = "isFromUI is " + strVal(eventInfo.isFromUI())
kaderEen.color = iif(eventInfo.isFromUI(), Blue, Red)

 ; sla de menuselectie van de gebruiker op in keuze
keuze = eventInfo.menuChoice()

 ; onderneem nu actie op basis van de menuselectie
switch
  case keuze = "&Knippen"     : action(EditCutSelection)
  case keuze = "&Verwijderen" : action(EditDeleteSelection)
```

```
  case keuze = "&Plakken"      : action(EditPaste)
endswitch

endmethod
```

Dit eenvoudige voorbeeld laat zien hoe MenuEvents en menuhandelingen met elkaar zijn verbonden. Het is van belang dat u deze relatie begrijpt als u complexere menu's voor applicaties wilt ontwikkelen.

HOOFDSTUK 14

# Weergavebeheer

Met weergavebeheer-objecten kunt u de vensters beheren die informatie presenteren aan een gebruiker. Als u weergavebeheer-objecten met de bijbehorende methodes gebruikt, kunt u het formaat, de vorm, de positie en de weergave besturen van de vensters waarmee de gebruiker werkt, zoals het bureaubladvenster, het formuliervenster, het rapportvenster en het tabelvenster.

In dit hoofdstuk worden de weergavebeheer-objecten van ObjectPAL behandeld (zie Tabel 8-1) en wordt beschreven hoe u de bijbehorende methodes gebruikt.

Tabel 14-1 Weergavebeheer-objecten

| Type | Beschrijving |
|---|---|
| Application | Het Bureaublad van Paradox |
| Form | Een Paradox-formulier dat in een eigen venster wordt weergegeven |
| Report | Een Paradox-rapport dat in een eigen venster wordt weergegeven |
| TableView | Een tabel die in een eigen venster wordt weergegeven |

## Application: het bureaubladvenster

Een Application-object verwijst naar het bureaubladvenster van de huidige Paradox-applicatie. Hoewel u meerdere applicaties tegelijkertijd kunt activeren, communiceren deze niet met elkaar en kunnen deze elkaar niet beïnvloeden.

U gebruikt Application-methodes om het bureaubladvenster te besturen. U kunt bijvoorbeeld een formulier verkleinen terwijl u het bureaublad het volledige formaat laat behouden. U kunt ook alle geopende Paradox-vensters verkleinen door een Application-variabele te gebruiken om het bureaublad te verkleinen. Methodes

van het Application-type zijn een subgroep van de methodes voor het Form-type (die gedetailleerd worden beschreven in de paragraaf "Form: een venster op gegevens", in dit hoofdstuk). U kunt deze methodes gebruiken om het formaat, de positie en de weergave van het applicatievenster te beheren, zoals in het volgende voorbeeld:

```
var ap Application endVar
ap.hide() ; maakt het bureaublad onzichtbaar
sleep(1000)
ap.show() ; maakt het bureaublad zichtbaar
sleep(1000)
ap.bringToTop() ; toont het bureaublad boven op alle andere vensters
ap.setPosition(100,100,2000,2100) ; stelt de linkerbovenhoek (100, 100),
                                    de breedte (2000) en de hoogte (2100) in
```

# Form: een venster op gegevens

Een Paradox-formulier speelt twee rollen:

- Weergavebeheer-object: een venster dat gegevens weergeeft. U kunt methodes van het Form-type gebruiken om een formulier te openen, weergave-attributen op te geven en het formulier te sluiten.
- Ontwerpobject: het hoogste niveau in de hiërarchie van ingesloten objecten. Een formulier heeft ingebouwde methodes die u kunt koppelen aan code en u kunt eigen code en variabelen koppelen om de methodes beschikbaar te maken voor alle objecten die het formulier bevat.

In deze paragraaf worden beide functies beschreven. Raadpleeg bovendien de paragraaf "UIObjecten" uit Hoofdstuk 13 voor het werken met ontwerpobjecten. In deze paragraaf worden ook manieren behandeld om dialoogvensters weer te geven, zoals het gebruik van formulieren als dialoogvensters en het gebruik van de dialoogvensters die in Paradox zijn ingebouwd. In deze paragraaf komen de volgende onderwerpen aan bod:

- Het formulier gebruiken als een weergavebeheer-object
- Werken met het gegevensmodel van een formulier
- Het formulier gebruiken als een ontwerpobject

## Het formulier als weergavebeheer-object

Als u een formulier als weergavebeheer-object wilt gebruiken, declareert u eerst een variabele van het Form-type en associeert u deze met een Paradox-formulier. Vervolgens kunt u de Form-variabele als een handle voor het formulier gebruiken. Met andere woorden: u kunt de Form-variabele in ObjectPAL-code gebruiken om het feitelijke formulier te manipuleren dat op het scherm wordt weergegeven.

Meestal wordt een Form-variabele met een Paradox-formulier geassocieerd via een **open**-instructie. De volgende code declareert bijvoorbeeld de Form-variabele *klantForm*. Met een **open**-instructie wordt de variabele vervolgens geassocieerd met een werkelijk formulier. Ten slotte wordt de variabele als een handle gebruikt om het formulier te manipuleren.

```
var
   klantForm Form              ; declareer de variabele
endVar

klantForm.open("klant.fsl")    ; associeer de variabele met een formulier
klantForm.minimize()           ; gebruik de variabele als een handle om het
formulier te manipuleren
```

Binnen een methode kunt u één Form-variabele associëren met meerdere formulieren, maar slechts met één formulier tegelijk. De volgende instructies openen bijvoorbeeld KLANT.FSL en ORDER.FSL en geven ORDER.FSL vervolgens als een pictogram weer:

```
var
   formVar Form
endVar

formVar.open("klant.fsl")
formVar.open("order.fsl")
formVar.minimize() ; verklein ORDER.FSL tot pictogram
```

Tabel 14-2 geeft een overzicht van veel gebruikte methodes van het Form-type. U gebruikt deze methodes als u een formulier als weergavebeheer-object gebruikt.

Tabel 14-2 Veel gebruikte methodes van het Form-type

| Methode | Beschrijving |
|---|---|
| create | Maakt een leeg formulier in een ontwerpvenster. |
| save | Slaat een formulier op. U kunt formulieren alleen in een ontwerpvenster opslaan. |
| load | Laadt een formulier van schijf in een ontwerpvenster. |
| run | Start een formulier (komt overeen met de optie 'FormulierlGegevens tonen'). |
| open | Opent en start een formulier |
| design | Schakelt over naar een ontwerpvenster (komt overeen met de optie 'FormulierlOntwerpen'). |
| isDesign | Geeft aan of het formulier zich in een ontwerpvenster bevindt. |

### *Tonen en verbergen*

Als u zelf wilt kunnen uitmaken of een formulier zichtbaar is voor de gebruiker, gebruikt u **show**, **hide** en **bringToTop**. In het volgende

voorbeeld is code gekoppeld aan het formulier *form0* dat twee andere formulieren bestuurt, *form1* en *form2*.

```
;deze code is gekoppeld aan form0
var
   form1, form2 Form      ; declareer 2 Form-variabelen om als handles te
gebruiken
endVar

form1.open("kaart.fsl")   ; toont form1
form2.open("steden.fsl")  ; toont form2 boven op form1

form1.bringToTop()        ; nu bevindt form1 zich bovenop
form1.hide()              ; form1 is onzichtbaar, form2 is zichtbaar
sleep(2000)
form1.show()              ; form1 is weer zichtbaar
bringloTop()              ; toont form0 bovenop
form1.close()             ; sluit form1
form2.close()             ; sluit form2

close()                   ; sluit form0 (het formulier waarin deze code wordt
uitgevoerd)
```

Zoals de bovenstaande code aangeeft, kunnen bepaalde methodes van het Form-type ook als procedures worden aangeroepen. Als u bijvoorbeeld **hide** wilt aanroepen als een methode, gebruikt u een Form-variabele als een handle voor een ander formulier, zoals in de volgende code:

```
var
   anderForm Form
endVar
anderForm.open("klant.fsl") ; gebruikt anderForm als een handle voor KLANT.FSL
anderForm.hide()
```

Als u **hide** als een procedure wilt aanroepen, gebruikt u geen handle: procedures van het Form-type werken op het huidige formulier (het formulier waarop de code wordt uitgevoerd). De volgende instructie verbergt bijvoorbeeld het huidige formulier:

```
hide()
```

### Een formulier sluiten

Als u klaar bent met een formulier, gebruikt u **close**. Bijvoorbeeld:

```
; Ga ervan uit dat deze methode is gekoppeld aan formEen
var formTwee endVar
formTwee.open("reisjes.fsl")

; doe hier uw verwerking,
; als u klaar bent, doe dan het volgende:
formTwee.close() ; sluit formTwee
close() ; sluit formEen
```

### Kenmerken instellen

In een ontwerpvenster kunt u Paradox interactief gebruiken om een groot aantal formulierkenmerken in te stellen, zoals de titel, de framestijl, de titelbalk en de schuifbalken. (Zie verder het *Handboek* voor meer informatie over het interactief instellen van kenmerken.) In ObjectPAL-code kunt u veel methodes van het Form-type gebruiken

voor het instellen van allerlei kenmerken en kunt u de puntnotatie gebruiken om rechtstreeks toegang te krijgen tot de kenmerken. U kunt bijvoorbeeld de methodes **getTitle** en **setTitle** van het Form-type gebruiken om de tekst op de titelbalk van het formuliervenster te manipuleren:

```
var
   mijnForm Form
   oudeTitel, nweTitel String
endVar
mijnForm.open("kaart.fsl")
mijnForm.getTitle(oudeTitel)
; plaatst huidige venstertitel in oudeTitel
nweTitel = "Ik heb de titel veranderd."
mijnForm.setTitle(nweTitel)
; plaatst "Ik heb de titel veranderd." op de titelbalk van het venster.
```

U kunt rechtstreeks toegang krijgen tot formulierkenmerken door de puntnotatie te gebruiken. Bijvoorbeeld:

```
var
   mijnForm  Form
   mijnPos, mijnFormaat Point
endVar
if mijnForm.open("order.fsl") then
   mijnPos = mijnForm.position          ; vraag het kenmerk 'Position' op
mijnFormaat = mijnForm.size             ; vraagt het kenmerk 'Size' op
mijnForm.title = "Orderformulier"       ; stelt het kenmerk 'Title' in
endIf
```

### Formaat bepalen

U kunt methodes van het Form-type niet alleen gebruiken om de kenmerken 'Size' en 'Position' in te stellen, maar ook om het formaat en de positie van een formulier te manipuleren. U kunt bijvoorbeeld met **maximize** een formulier het formaat van het hele scherm geven, het formulier tot een pictogram verkleinen met **minimize** of het formulier een bepaalde positie geven met **setPosition**. Als u informatie wilt over het formaat van het formulier, gebruikt u **isMaximized** en **isMinimized**. Stel dat de volgende code aan een formulier is gekoppeld:

```
; deze methode is gekoppeld aan een ander formulier, niet aan proteus
var
   proteus Form
endVar
proteus.open("kaart.fsl") ; leest formulier van schijf
proteus.minimize() ; verkleint het tot een pictogram
proteus.maximize() ; geeft formulier het formaat van het hele scherm
if proteus.isMaximized() then
   proteus.setPosition(20,20,6000,5000)
;  stelt linkerbovenhoek, breedte en hoogte in (in twips)
endIf
```

Deze code declareert de Form-variabele *proteus* om deze als een handle te gebruiken. De variabele *proteus* wordt dus in de code gebruikt om het formuliervenster op het scherm te manipuleren. De aanroep **setPosition** gebruikt argumenten die de coördinaten van de linkerbovenhoek, de breedte en de hoogte opgeven. Alle waarden die

betrekking hebben op het scherm, worden uitgedrukt in *twips*. Een twip is een maateenheid die gelijk is aan 1/1440 van een inch (1/20 van een printerpunt).

Zie "Multi-formulier applicaties", verderop in dit hoofdstuk, voor meer informatie over formulierkenmerken.

## Gegevensmodel van een formulier

U kunt ObjectPAL gebruiken om het gegevensmodel van een formulier te manipuleren door te vragen en te bepalen met welke tabel of tabellen het formulier is verbonden. De methodes uit het overzicht van Tabel 14-3 worden gedetailleerd beschreven in de online ObjectPAL Help.

Tabel 14-3 Methodes voor het werken met het gegevensmodel van een formulier

| Methode | Beschrijving |
|---|---|
| dmAddTable | Voegt een tabel toe aan het gegevensmodel van een formulier |
| dmGet | Haalt een waarde uit een veld van een tabel in het gegevensmodel van het formulier |
| dmHasTable | Geeft aan of een opgegeven tabel zich in het gegevensmodel van het formulier bevindt |
| dmPut | Wijst een waarde toe aan een veld van een tabel in het gegevensmodel van het formulier |
| dmRemoveTable | Verwijdert een tabel uit het gegevensmodel van een formulier |

Met deze methodes kunt u rechtstreeks waarden veranderen in een tabel van het gegevensmodel, zonder een koppeling te maken met een bepaald ontwerpobject op het formulier.

De tabelnaam die u opgeeft, moet overeenkomen met de naam van een tabel in het huidige gegevensmodel, met inbegrip van de alias. Als u ":MIJNDB:MIJNDATA.DB" in het dialoogvenster 'Gegevensmodel' hebt gekoppeld, geeft u ":MIJNDB:MIJNDATA.DB" op als argument voor **dmGet** of **dmPut**. De veldnaam is de fysieke veldnaam in de tabel. Deze is niet verbonden met de naam van een ontwerpobject. De volgende instructies halen bijvoorbeeld de waarde uit het veld 'Aantal' in de tabel *Antwrd* in de privé-directory van de gebruiker en wijzen deze waarde toe aan de variabele *antwrdAantal*:

```
var antwrdAantal AnyType endVar
dmGet(":PRIV:ANTWRD.DB", "Aantal", antwrdAantal)
```

### *Directorypaden en het gegevensmodel*

Als u een tabel op een interactieve manier toevoegt aan het gegevensmodel van een ontwerpvenster, slaat de tabel de naam als volgt op:

- Tabellen met absolute paden of aliassen, zoals ":FRITS:VERKOOP.DB" of "C:\TABELLEN\ORDER.DB", slaan het pad of de alias precies op zoals deze zijn opgegeven.
- Tabellen met ":WORK:" laten die alias weg en slaan alleen de basisnaam van de tabel op.

Als u een formulier opent, worden de tabellen uit het gegevensmodel als volgt gevonden:

- Tabellen met padnamen of aliassen gebruiken de volledige padnaam.
- Tabellen zonder pad of alias (dus met alleen de basisnaam) gebruiken het pad of de alias van de plaats waar het formulier zich bevindt.

Als u een formulier opent in C:\TEMP (terwijl uw werkdirectory I:\STARTEN is) en het gegevensmodel van dat formulier BKBESTEL.DB bevat, is het pad voor de tabel C:\TEMP\ BKBESTEL.DB.

Als de tabel een relatief pad heeft, zoals MIJNDIR\BKBESTEL.DB, interpreteert het formulier het pad als C:\TEMP\MIJNDIR\ BKBESTEL.DB. Relatieve paden werken ook met aliassen. De alias :ALIAS:MIJNDIR\BKBESTEL.DB is dus ook een geldig pad.

***Opmerking*** Het formulier zoekt niet naar tabellen. Als de tabellen zich niet op de plaats bevinden, waar het formulier deze verwacht, geeft Paradox een foutmelding.

Als het formulier een tabel vindt, wordt hetzelfde algoritme toegepast op alle opzoektabellen die deze tabel gebruikt. Als de opzoektabel in het bestand van de validiteitscontrole een absoluut pad heeft, wordt dit pad gebruikt. Als de opzoektabel geen pad heeft, wordt ervan uitgegaan dat de opzoektabel zich op dezelfde plaats bevindt als de tabel zelf. Als in het bovenstaande voorbeeld de opzoektabel bijvoorbeeld "MIJNOPZK.DB" is, verwacht het formulier dat de tabel zich bevindt in C:\TEMP\MIJNDIR\MIJNOPZK.DB.

Als u een formulier opslaat in een andere directory dan de werkdirectory, wordt dezelfde strategie gebruikt om volledige namen toe te wijzen aan de tabellen.

## Formulier als ontwerpobject

Het formulier kan ook als een ontwerpobject fungeren. Als zodanig is het formulier het hoogste niveau in de hiërarchie van ingesloten objecten. Het sluit namelijk alle andere objecten in. Code die aan een formulier is gekoppeld, is zichtbaar voor alle objecten die het formulier insluit.

**Tip** Koppel eigen methodes en procedures die u veel gebruikt, aan een formulier. Dit is efficiënter dan de methode te kopiëren naar alle objecten die de methode aanroepen.

In de onderwerpen van deze paragraaf wordt beschreven hoe u code koppelt aan een formulier en hoe u de kenmerken van ontwerpobjecten in andere formulieren instelt. Raadpleeg Hoofdstuk 13 voor meer informatie over ontwerpobjecten en de hiërarchie van ingesloten objecten.

### Methodes bewerken

Net als de methodes van een knop kunt u methodes van een formulier koppelen en bewerken. U gaat als volgt te werk:

1. Kies 'Kenmerken | Formulier | Methodes' in het formulierontwerpvenster om het methodevenster te openen. (Snelle manier: druk een aantal malen op *Esc*, totdat er geen objecten meer zijn geselecteerd. Druk vervolgens op *Ctrl-Spatiebalk* om het methodevenster te openen.)
2. Selecteer een of meer ingebouwde methodes of koppel eigen code (bijvoorbeeld een eigen methode of procedure), zoals u dat voor elk ander object zou doen.

Het Form-type bevat ook methodes die u kunt gebruiken om te bepalen hoe een formulier wordt weergegeven.

### Kenmerken van andere objecten instellen

U kunt de hiërarchie van ingesloten objecten gebruiken om kenmerken van objecten op andere formulieren te manipuleren. Stel dat code in *formEen formTwee* opent en dat *formTwee* een tekstvak (*deTekst*) en een ellips (*deEllips*) insluit. Methodes in *formEen* kunnen kenmerken instellen van *deTekst* en *deEllips* door het pad van ingesloten objecten op te geven. Bijvoorbeeld:

```
; deze code is gekoppeld aan formEen
var
   formTwee Form
endVar
formTwee.open("order.fsl")
formTwee.deTekst.text = "Deze tekst is verstuurd door formEen."
formTwee.deEllips.color = "Red"
```

# Multi-formulier applicaties

Multi-formulier applicaties zijn applicaties die meer dan één formulier gebruiken (na elkaar of tegelijkertijd.) Multi-formulier applicaties worden gemaakt van standaardformulieren en dialoogvensters, die een speciaal type formulier zijn. Dialoogvensters worden meestal gebruikt om in een bepaalde volgorde gegevens uit te wisselen met de gebruiker, bijvoorbeeld om een wachtwoord te

vragen voordat de gebruiker verder kan gaan met de bewerking. Meervoudige standaardformulieren worden daarentegen vooral gebruikt om een applicatie onder te verdelen in functionele modules.

In deze paragraaf worden de basisconcepten voor het maken van multi-formulier applicaties beschreven. De volgende onderwerpen worden behandeld:

- Verschillen tussen formulieren en dialoogvensters
- Een dialoogvenster ontwerpen
- Multi-venster interactie
- Van tevoren ontworpen dialoogvensters gebruiken
- Geavanceerde onderwerpen, zoals **openAsDialog** en het kenmerk 'DeskTopForm'.

## Verschillen tussen formulieren en dialoogvensters

Er bestaat een aantal kenmerkende verschillen tussen Paradox-formulieren en Paradox-dialoogvensters.

*Modaliteit*

Vensters kunnen modaal of niet-modaal zijn. Een *modaal* venster is een venster met exclusieve rechten op focus en acties. Een *niet-modaal* venster daarentegen staat de gebruiker toe zich vrij te verplaatsen tussen dit venster en andere vensters, toegang te krijgen tot systeemmenu's en de TurboBalk en andere Windows-applicaties aan te roepen. Paradox kent ook een tussenliggende status, waarbij alleen aan het wachtende formulier focus wordt geweigerd.

Paradox kent drie soorten vensters:

- *Standaardformulieren* zijn altijd niet-modaal. Een formulier gedraagt zich echter alsof het tijdelijk is onderbroken als het op een ander formulier of dialoogvenster wacht.
- *Niet-modale dialoogvensters* gedragen zich in veel opzichten als formulieren. Het formulier dat het niet-modale dialoogvenster aanroept, wordt tijdelijk onderbroken zolang het moet wachten, maar de gebruiker heeft ondertussen toegang tot andere formulieren, menu's enzovoort.
- *Modale dialoogvensters* gedragen zich op een strikt modale manier zolang erop moet worden gewacht of als deze *interactief zijn aangeroepen*. Anders gedragen modale dialoogvensters zich als niet-modale dialoogvensters.

Het verschil tussen modale en niet-modale dialoogvensters is subtiel. De dialoogvensters gedragen zich hetzelfde als er niet op wordt gewacht. Als er wel op wordt gewacht, behoudt een modaal dialoogvenster een strikt modale focus. Een modaal dialoogvenster

wordt vooral gebruikt als een programma aanvullende gegevens nodig heeft voordat het kan worden vervolgd. De gebruiker kan geen andere bewerking verrichten, tenzij hij de opdracht annuleert of aanvullende gegevens verstrekt.

Een niet-modaal dialoogvenster, daarentegen, geeft de gebruiker toegang tot de systeemmenu's en andere vensters. Een goed voorbeeld hiervan is een opdracht voor het zoeken van tekst: het dialoogvenster blijft geopend terwijl de zoekopdracht wordt uitgevoerd. De gebruiker kan daarna naar het dialoogvenster terugkeren en dezelfde tekst of een andere tekst zoeken.

*Positie*

De positiecoördinaten van een dialoogvenster worden berekend ten opzichte van het scherm en de positiecoördinaten van een formulier worden berekend ten opzichte van het bureaublad. In beide gevallen heeft de oorsprong van het raster de coördinaten (0,0).

*Opmerking voor ervaren Windows-programmeurs*

De positiecoördinaten van dialoogvensters worden berekend ten opzichte van het scherm omdat dialoogvensters *geen* MDI-subelementen zijn (zie "MDI-subelement-vensters", verderop in deze paragraaf.

*Altijd bovenop*

Dialoogvensters worden altijd boven op andere vensters weergegeven. Het ene dialoogvenster kan een ander dialoogvenster bedekken, terwijl een normaal formulier nooit een dialoogformulier kan bedekken. Formulieren kunnen vanuit andere formulieren bovenop worden geplaatst (zie de **bringToTop**-methode van het Form-type).

*Formaat en andere weergavekenmerken tijdens uitvoering*

Het formaat van formulieren kan worden aangepast tijdens de uitvoering. Het formaat van dialoogvensters kan niet interactief worden aangepast.

Zowel formulieren als dialoogvensters kunnen een horizontale en een verticale schuifbalk bevatten. Formulieren moeten een titelbalk bevatten waarmee de gebruiker het venster vrijelijk over het scherm kan verplaatsen, terwijl een titelbalk bij dialoogvensters optioneel is.

*Verplaatsing buiten het bureaubladvenster*

Zowel modale als niet-modale dialoogvensters kunnen buiten het bureaubladvenster worden geplaatst. Normale formulieren, die MDI-subelementen zijn, moeten binnen de grenzen van het hoofdbureaublad blijven.

*Menu's manipuleren*

Dialoogvensters kunnen niet communiceren met het menu. Dialoogvensters kunnen de menubalk niet veranderen en accepteren geen menu-opties. Gebruik daarom geen dialoogvenster als de gebruiker moet kunnen communiceren met menu's.

Als een modaal dialoogvenster actief is, kan de gebruiker niet communiceren met een ander deel van Paradox (andere formulieren of tabelvensters) terwijl er op het dialoogvenster wordt gewacht.

*MDI-subelement*

Voor ervaren Windows-programmeurs kan het verschil tussen Paradox-formulieren en Paradox-dialoogvensters als volgt worden samengevat: een formulier is een MDI-subelement van Windows terwijl een dialoogvenster een pop-up venster is. De consequenties van dit verschil worden uitgelegd in de paragraaf over geavanceerde onderwerpen.

## Dialoogvensters ontwerpen

Als u een dialoogvenster ontwerpt, gaat u als volgt te werk: kies 'Bestand | Nieuw | Formulier' om een nieuw formulier te maken. Stel vervolgens de vensterstijl van het formulier in op *dialoogvenster*: kies 'Kenmerken | Formulier | Vensterstijl aanpassen' om het venster 'Kenmerken formuliervenster' te openen. U kunt nu de vensterstijl, de vensterkenmerken, de framekenmerken en de kenmerken van de titelbalk instellen.

Kies 'Dialoogvenster' in het paneel 'Vensterstijl' om het formulier de attributen van een dialoogvenster te geven. Hierdoor worden de opties voor de titelbalk(verplicht bij standaardformulieren) en het kenmerk 'Modaal' beschikbaar.

Als u de kenmerken hebt ingesteld, slaat u het formulier op met 'Bestand | Opslaan als'. De kenmerkinstellingen worden nu bewaard. Als u het formulier een volgende keer opent, gedraagt het zich als een dialoogvenster. Zie verder het *Handboek* voor meer informatie over het interactief instellen van formulierkenmerken.

***Opmerking***

Ontwerp een dialoogvenster zorgvuldig. Laat altijd een mogelijkheid open om het dialoogvenster te sluiten. Als een modaal dialoogvenster actief is, moet de gebruiker dit venster altijd kunnen verlaten.

*Kenmerken instellen met ObjectPAL*

De kenmerken van formulieren en dialoogvensters kunnen ook met de **open**-methode van ObjectPAL worden ingesteld als het venster is geopend. Het is zeer aan te bevelen dat u de kenmerken van een dialoogvenster instelt met het paneel 'Vensterstijl' in het formulier-ontwerpvenster. Het is gemakkelijker de kenmerken interactief in te stellen als u het formulier ontwerpt en daarna **open** te gebruiken, dan uit te zoeken hoe u een goede en consistente verzameling vensterstijlconstanten beheert.

*Positie en formaat opgeven*

Standaardformulieren kunnen worden ingesteld met het kenmerk 'Passend maken', zodat het formaat van het venster wordt aangepast aan het omringende pagina-object. U kunt het formaat van een formulier dus interactief aanpassen door de afmetingen van de pagina vooraf in te stellen. Anders wordt het standaard MDI-vensterformaat gebruikt en moet u het formaat misschien interactief aanpassen.

U kunt het standaardformaat ook veranderen door **setPosition** aan te roepen of door een formaat op te geven in de **open**-instructie. De **open**-instructie in de volgende code stelt bijvoorbeeld de

linkerbovenhoek van het formulier in op (100, 100), de breedte op 5000 twips en de hoogte op 2000 twips.

```
var
   ordForm Form
endVar
ordForm.open("order.fsl", WinStyleDefault, 100, 100, 5000, 2000)
```

Er kan zich een situatie voordoen waarin u niet alle vier de positiecoördinaten wilt opgeven, bijvoorbeeld als u het formulier één inch onder de standaardpositie onder de TurboBalk wilt weergeven. Gebruik in dat geval als volgt de constante WinDefaultSize:

```
var
   ordForm Form
endVar
ordForm.open("order.fsl", WinStyleDefault, WinDefaultCoordinate, 1440,
WinDefaultCoordinate, WinDefaultCoordinate)
```

De constante WinDefaultCoordinate fungeert als plaatshouder en wordt vervangen door de standaardwaarde (formaat of positie) van het formulier als u het formulier start.

*Modale dialoogvensters*

Vergeet niet dat als u het kenmerk 'Dialoogvenster' op True instelt, u het kenmerk 'Modaal' expliciet moet instellen om een modaal dialoogvenster te krijgen.

## Multi-venster interactie

Er zijn veel verschillende soorten multi-venster interacties. U kunt bijvoorbeeld de volgende dingen doen:

- Een modaal dialoogvenster openen, communiceren met de gebruiker, het antwoord van de gebruiker teruggeven (aan het aanroepend formulier) en vervolgens het dialoogvenster sluiten
- Een niet-modaal dialoogvenster gebruiken, zodat de gebruiker gegevens of handelingen kan invoeren die worden herhaald
- Twee of meer formulieren gebruiken in een applicatie

Als u variabelen of objecten op een ander formulier wilt besturen, gebruikt u de puntnotatie om toegang te krijgen tot de objecten, waarden en kenmerken die u nodig hebt. Dit vormt de basis voor multi-venster programmering. Verscheidene ingebouwde methodes zorgen ervoor dat de coördinatie gemakkelijker verloopt.

### *wait, formReturn, close*

Als u een dialoogvenster opent, wilt u dat de gebruiker antwoordt. U wilt bijvoorbeeld dat de gebruiker bepaalde gegevens invoert of een bericht ziet. Terwijl u wacht, moet de code ook wachten, omdat de volgende instructies mogelijk een of meer waarden nodig hebben die het dialoogvenster teruggeeft. U gebruikt in zo'n geval een **wait**-instructie om de uitvoering tijdelijk uit te stellen, totdat het aangeroepen formulier de besturing teruggeeft aan het aanroepend formulier.

Het aangeroepen formulier of het aangeroepen dialoogvenster doet dit met een aanroep van **formReturn** of **close**. Er zijn drie manieren waarop u de besturing vanuit een dialoogvenster kunt teruggeven aan een aanroepend formulier:

- Gebruik de menu's van het dialoogvenster om het interactief te sluiten.
- Roep **close** aan.
- Roep **formReturn** aan.

*Dialoogvenster interactief sluiten*

Als een dialoogvenster een Systeemmenu heeft, kunt u het venster interactief sluiten door 'Sluiten' te kiezen in het Systeemmenu of door te drukken op *Ctrl-F4*. Beide handelingen geven de besturing terug aan het aanroepend formulier en geven de logische waarde False terug.

Als de gebruiker klaar is in een dialoogvenster, vraagt u de waarden op die u nodig hebt, en gebruikt u **close** om het dialoogvenster te sluiten. Als u een dialoogvenster sluit, kunt u niet meer naar de objecten, waarden of kenmerken van dat venster verwijzen, tenzij u het venster opnieuw opent.

*close aanroepen*

U gebruikt de **close**-methode om het dialoogvenster (of het standaardformulier) te sluiten en de besturing terug te geven aan het aanroepend formulier. U kunt een optioneel argument opnemen in de **close**-instructie om een waarde terug te geven aan het aanroepend formulier. Als u **close** zonder argument aanroept, geeft deze methode de logische waarde False terug.

*formReturn aanroepen*

De **formReturn**-methode geeft de besturing terug aan een aanroepend formulier dat tijdelijk is onderbroken door een **wait**-instructie. In tegenstelling tot bij **close**, blijft het dialoogvenster of formulier dat **formReturn** aanroept, geopend. De **formReturn**-instructie kan een waarde teruggeven aan het aanroepend formulier. Zonder argument geeft deze instructie de logische waarde False terug.

Het aanroepend formulier kan een **wait**-lus instellen, zodat het aangeroepen dialoogvenster **formReturn** vaker kan aanroepen. Het aangeroepen formulier kan bijvoorbeeld de teruggegeven waarde controleren en **wait** blijven aanroepen voor het formulier totdat er een geldige waarde wordt teruggegeven.

*Modale dialoogvensters*

Als het aangeroepen dialoogvenster modaal is, kan de gebruiker niet met iets anders communiceren zolang de aanroeper (of een ander formulier) op dit venster wacht. Als het dialoogvenster **formReturn** heeft aangeroepen, kan de focus worden teruggegeven aan andere vensters.

*wait-instructies nesten*

U kunt **wait**-instructies nesten, maar u moet er dan voor zorgen dat u de besturing teruggeeft in een volgorde die omgekeerd is aan de volgorde waarin u de **wait**-instructies hebt aangeroepen. Het

formulier *formA* kan bijvoorbeeld *formB* openen en **wait** aanroepen. Vervolgens kan *formB formC* openen en **wait** aanroepen. —*formA* en *formB* reageren dan niet op acties totdat *formC* de besturing teruggeeft. Als u de besturing teruggeeft, vergeet dan niet dat deze eerst teruggaat van *formC* naar *formB* en daarna van *formB* naar *formA*.

### Voorbeeld

Dit voorbeeld demonstreert het gebruik van **close**, **formReturn** en **wait**, en geeft bovendien een voorbeeld van rechtstreekse toegang tot objecten op andere formulieren. Zie verder de MAST-applicatie, die verschillende formulieren en dialoogvensters gebruikt.

In dit voorbeeld ziet u code die is gekoppeld aan twee formulieren. Het eerste formulier is het hoofdformulier en bevat een knop die een tweede formulier opent. De activiteit in het eerste formulier wordt tijdelijk onderbroken totdat het tweede formulier de besturing teruggeeft. Als de gebruiker de besturing teruggeeft door op de knop 'OK' te klikken, wordt code uitgevoerd die de waarde leest van *AntwrdVeld*, een veldobject in het dialoogvenster dat een waarde moet bevatten die door de gebruiker is ingevoerd. Deze waarde wordt als een argument doorgegeven aan de eigen procedure **doeIets** (ergens anders gedefinieerd). Als de gebruiker de besturing teruggeeft door op de knop 'Annuleren' te klikken, sluit het dialoogvenster zichzelf.

Deze code is gekoppeld aan een knop op het hoofdformulier:

```
method pushButton(var eventInfo Event)   ; Hoofdformulier (aanroepend)
var
   dlgForm Form
   dlgTerug AnyType
endVar

if dlgForm.openAsDialog("myDlg.fsl", WinStyleDialogFrame, 100, 100, 2880, 2880)
then
   dlgTerug = dlgForm.wait()  ; tijdelijk onderbreken totdat dialoogvenster
                              ; besturing teruggeeft
   switch
      case dlgTerug = "OK" :
         doeIets(dlgForm.AntwrdVeld.Value) ; dialoogvenster is nog open
      case dlgTerug = "Cancel" :
         return                                   ; dialoogvenster heeft zichzelf
                                                  ; gesloten
   endSwitch
   dlgForm.close()                                ; sluit dialoogvenster endif
endmethod
```

Het tweede formulier, dat als een dialoogvenster is geopend, bevat twee knoppen, 'OK' en 'Annuleren'. De volgende code is gekoppeld aan de knop 'OK'. Deze code geeft de besturing terug aan het hoofdformulier en geeft de waarde "OK" terug aan de **wait**-instructie van het hoofdformulier.

```
method pushButton(var eventInfo Event)   ; code knop 'OK'
   formReturn("OK")       ; geeft waarde terug, sluit dit dialoogvenster niet
endmethod
```

De volgende code is gekoppeld aan de knop 'Annuleren' van het tweede formulier. Deze code gebruikt een **close**-instructie om het dialoogvenster te sluiten en geeft "Cancel" terug aan de **wait**-instructie van het hoofdformulier.

```
method pushButton(var eventInfo Event)   ; code knop 'Annuleren'
close("Cancel")          ; geef een waarde terug en sluit dit dialoogvenster
endmethod
```

*Modaal*

Als een ObjectPAL-instructie een formulier opent waarvan het kenmerk 'Modaal' is ingesteld op True, wordt de uitvoering niet tijdelijk onderbroken door de methode. De volgende code opent bijvoorbeeld MODFORM.FSL, een formulier waarvan het kenmerk 'Modaal' interactief is ingesteld op True. Na de **open**-instructie wordt een eenvoudige tellus uitgevoerd die een bericht weergeeft.

```
var
   modForm Form
endVar
modForm.open("modform.fsl") ; open formulier
for i from 1 to 5 ; voer lus uit
   message(i)
endFor
```

De lus wordt meteen na de **open**-instructie uitgevoerd. De uitvoering wordt alleen tijdelijk onderbroken door de **wait**-instructie.

## Voorgedefinieerde dialoogvensters

ObjectPAL bevat verscheidene voorgedefinieerde modale dialoogvensters. Methodes voor het openen van deze dialoogvensters zijn gedefinieerd voor het System-type en beginnen met de letters "msg" (bijvoorbeeld **msgYesNoCancel**). Raadpleeg de online ObjectPAL Help voor een volledig overzicht.

```
method pushButton(var eventInfo Event)
   msgYesNoCancel("Amuseren we ons al?", "Wij zijn benieuwd.")
endmethod
```

Veel van deze dialoogvensters geven een waarde terug. U kunt deze dialoogvensters dus gebruiken om gebruikersinvoer te krijgen. De volgende code declareert bijvoorbeeld de variabele *deOptie*, waaraan een waarde wordt toegewezen, afhankelijk van de knop waarop de gebruiker klikt om het dialoogvenster te sluiten. Vervolgens bepaalt een **switch**...**endSwitch**-structuur op basis van de waarde van *deOptie* wat er vervolgens moet gebeuren.

```
method pushButton(var eventInfo Event)
   var
      deOptie String
   endVar

   deOptie = msgYesNoCancel("Afsluiten", "Wilt u werkelijk afsluiten?")
   switch
      case deOptie = "Yes" : doExit() ; voer een eigen methode uit
      otherwise            : return
   endSwitch
endmethod
```

Deze dialoogvensters zijn strikt modaal en de uitvoering van de code wordt tijdelijk onderbroken totdat de gebruiker antwoordt. Er bestaat dus een impliciete **wait**-instructie van het systeem. In de volgende code wordt de tellus niet uitgevoerd voordat de gebruiker het dialoogvenster beantwoordt door 'OK' te kiezen of het systeemmenu te gebruiken om het venster te sluiten.

```
msgInfo("Modaal dialoogvenster.", "Kies 'OK' om de tellus te starten.")
for i from 1 to 5 ; Deze lus wordt niet uitgevoerd voordat
   message(i)     ; de gebruiker het dialoogvenster beantwoordt. endFor
```

Zie verder Hoofdstuk 17 voor meer informatie over methodes van het System-type.

## Paradox-dialoogvensters aanroepen

U kunt ObjectPAL gebruiken om een groot gedeelte van de ingebouwde dialoogvensters van Paradox te openen. De procedures beginnen met de letters "dlg" (bijvoorbeeld **dlgCreate**) en zijn gedefinieerd voor het System-type. Zie de online ObjectPAL Help voor gedetailleerdere informatie over specifieke procedures.

De volgende instructie opent bijvoorbeeld het Paradox-dialoogvenster voor het maken van de tabel ORDER.DB, alsof u 'Bestand | Nieuw | Tabel' hebt gekozen:

```
dlgCreate("order.db")
```

***Opmerking*** Als deze dialoogvensters eenmaal zijn geopend, hebt u er als programmeur geen controle over. De gebruiker moet er dus zelf voor zorgen dat hij de ingebouwde dialoogvensters correct gebruikt.

## Geavanceerde onderwerpen

Deze paragraaf bevat informatie over formulieren en dialoogvensters die interessant kunnen zijn voor de ervaren programmeur.

### *MDI-subelement-vensters*

Als u het verschil tussen formulieren en dialoogvensters wilt begrijpen, is het nuttig (hoewel niet noodzakelijk) het Windows-concept *MDI-subelement* te kennen. In Windows-termen is Paradox een *MDI*-applicatie. De Multiple Document Interface (MDI) is een standaardinterface voor Windows-applicaties die het de gebruiker mogelijk maakt in meerdere geopende documenten te werken. U kunt een MDI-applicatie beschouwen als een mini-sessie van Windows, inclusief applicaties die door vensters of pictogrammen worden vertegenwoordigd.

Het bureaublad is een *MDI-framevenster*. Dat wil zeggen dat het achter de schermen alle formulier- en actiehandelingen van uw applicaties beheert. Een standaardformulier is een MDI-subelement-venster. Een MDI-subelement heeft de volgende eigenschappen:

- Het kan worden vergroot tot het volledige formaat van het framevenster of worden verkleind tot een pictogram in het framevenster.
- Het verschijnt nooit buiten het kader van het framevenster.
- Het heeft geen menu. De functies worden daarom bestuurd door het menu van het framevenster. (Met de ObjectPAL-methodes kunt u echter wel de menubalk en de TurboBalk aanpassen en menuhandelingen van de gebruiker onderscheppen.)
- De titel van een MDI-subelement-venster is vaak hetzelfde als de naam van het geopende bestand dat met dat venster is geassocieerd (ook hier onder optioneel beheer van ObjectPAL).
- Het venster kan naast andere vensters worden geplaatst, trapsgewijs worden weergegeven en handmatig worden vergroot of verkleind door de gebruiker.

### *Standaardmenu*

Het dialoogvenster 'Kenmerken formuliervenster' bevat het aankruisvak 'Standaardmenu'. Dit aankruisvak is alleen beschikbaar voor standaardformulieren. Als het aankruisvak is geselecteerd (de standaardinstelling), wordt er een formulier gestart dat het normale runtime menu weergeeft. In bepaalde applicaties kunnen menu's hierdoor gaan "knipperen".

Als 'Standaardmenu' niet is geselecteerd, worden de menu's met rust gelaten als er een formulier wordt gestart, tenzij het formulier ObjectPAL-code bevat die menu's maakt. Dat wil zeggen dat het formulier het menu weergeeft dat actief was toen het formulier werd gestart. Dit kan knippering van het menu helpen voorkomen.

### *openAsDialog*

**openAsDialog** is verwant met de Form-methode **open**. Deze methode is alleen bedoeld voor ervaren programmeurs (bijvoorbeeld om aangemaakte formulieren met specifieke kenmerken te starten). **openAsDialog** verschilt op twee belangrijke punten van **open**:

- U kunt elke stijlconstante van Windows gebruiken om weergave-attributen op te geven. (Met **open** kunt alleen geselecteerde constanten gebruiken.)
- Het dialoogvensterkenmerk van een formulier dat is geopend met **openAsDialog**, is ingesteld op True.

**open** en **openAsDialog** gebruiken argumenten om de positie, het formaat en de weergave-attributen van het formulier aan te geven. Tabel 14-4 geeft een overzicht van de stijlconstanten van Windows die u kunt gebruiken met deze methodes. Constanten worden beschreven in de online ObjectPAL Help.

Tabel 14-4 Constanten voor 'open' en 'openAsDialog'

| openAsDialog | open |
|---|---|
| WinDefaultCoordinate | WinDefaultCoordinate |
| WinStyleBorder | |
| WinStyleControlMenu | |
| WinStyleDefault | WinStyleDefault |
| WinStyleDialog | |
| WinStyleDialogFrame | |
| WinStyleHidden | WinStyleHidden |
| WinStyleHScroll | WinStyleHScroll |
| WinStyleMaximize | WinStyleMaximize |
| WinStyleMaximizeButton | |
| WinStyleMinimize | WinStyleMinimize |
| WinStyleMinimizeButton | |
| WinStyleModal | |
| WinStyleThickFrame | |
| WinStyleTitleBar | |
| WinStyleVScroll | WinStyleVScroll |

*Opmerking over kenmerken*

Sommige kenmerken kunnen niet interactief worden ingesteld en moeten daarom met ObjectPAL op het formulier worden ingesteld met **open** of **openAsDialog**. U opent een verborgen, verkleind of vergroot formulier respectievelijk met de kenmerken WinStyleHidden, WinStyleMinimize en WinStyleMaximize.

***Opmerking***

Een aanroep van **openAsDialog** maakt een formulier niet standaard modaal. De methode stelt alleen de opgegeven weergave-attributen in. Als u met deze methode een dialoogvenster wilt maken, gebruikt u de constante WinStyleModal.

## DesktopForm

DesktopForm is een lezen/schrijven kenmerk van formulierobjecten dat de applicatie een achtergrondformulier geeft voor de behandeling van menu-acties als er geen andere formulieren actief zijn op het bureaublad. Dit kenmerk moet voor slechts één formulier op het bureaublad tegelijk zijn ingesteld.

Het kenmerk heeft alleen effect als het bureaublad geen andere MDI-subelement-vensters bevat. In dit geval worden menu-acties naar dit formulier gestuurd (via de **menuAction**-methode), zoals u ziet in het volgende codefragment:

```
method open(var eventInfo Event)
   var
      f Form
   endVar
   f.attach()
```

```
   f.DesktopForm = TRUE
   f.hide()
endMethod
```

Deze code laat het formulier actief, maar verborgen op het bureaublad. Het formulier kan op menu-acties reageren als het bureaublad leeg wordt. Als er een ander formulier wordt geopend, worden menu-acties standaard weer naar dat nieuwe formulier worden gestuurd. Als u wilt dat het nieuwe formulier het menu van het bureaublad gebruikt, stelt u het kenmerk 'StandardMenu' van het nieuwe formulier in op False. U moet nog wel code koppelen aan de ingebouwde **menuAction**-methode van het formulier.

DesktopForm verschijnt niet in het dialoogvenster 'Kenmerken formuliervenster'. Het kenmerk is alleen beschikbaar vanuit ObjectPAL. Het is een blijvend kenmerk. Als een formulier wordt opgeslagen en dit kenmerk is ingesteld op True, is het kenmerk de volgende keer dat het formulier wordt geopend, ook True. Het maakt hierbij niet uit of het formulier interactief of met ObjectPAL wordt geopend.

# Report: gegevens opmaken en afdrukken

Met variabelen en methodes van het Report-type kunt u een rapportvenster besturen. Een Report-variabele is een handle naar een rapport waarmee u het formaat, de positie en de weergave van het rapportvenster kunt veranderen, een voorbeeld van een rapport kunt tonen en een rapport kunt afdrukken.

U gebruikt **load** om een rapportbestand in een ontwerpvenster te laden, **open** om het rapport te openen in de Gegevens tonen-modus en **print** om een rapport te openen en af te drukken. U kunt geen methodes koppelen aan objecten in een rapport.

Het volgende voorbeeld demonstreert een eenvoudige manier om een rapport vanaf een schijf af te drukken. Het rapport uit het voorbeeld is van tevoren ontworpen en opgeslagen.

```
method pushButton(var eventInfo Event)
   var
      klantRpt Report
   endVar

   klantRpt.open("klant.rdl")
   klantRpt.print()
endmethod
```

Als deze methode wordt uitgevoerd, opent de **open**-instructie het rapport en initieert de **print**-instructie de afdrukprocedure. Eerst wordt het Paradox-dialoogvenster 'Bestand afdrukken' geopend. In dit dialoogvenster kunt u het aantal pagina's opgeven dat u wilt

afdrukken. U moet dit dialoogvenster sluiten om het rapport af te drukken.

Als u meer controle wilt over de manier waarop een rapport wordt afgedrukt, kunt u ReportPrintInfo gebruiken, een voorgedefinieerd record met de volgende structuur:

```
name            String   ; start dit rapport als het nog niet open is
masterTable     String   ; hoofdtabelnaam
queryString     String   ; start deze query (feitelijke query-reeks)
restartOptions  SmallInt ; wat moet er gebeuren als gegevens worden gewijzigd
                         ; tijdens afdrukken?
printBackwards  Logical  ; vooruit FALSE, achteruit TRUE, standaard False
makeCopies      Logical  ; Wie maakt kopieën: Paradox of printer?
                         ; Indien True: Paradox maakt kopieën
panelOptions    SmallInt
nCopies         SmallInt ; aantal kopieën, standaard is 1
startPage       LongInt  ; eerste pagina, standaard is 1
endPage         LongInt  ; laatste pagina: eindpagina
pageIncrement   SmallInt ; stapgrootte voor afdrukken in meerdere rondes,
                         ; standaard is 1
xOffset         LongInt  ; horizontale marge
yOffset         LongInt  ; verticale marge
orient          SmallInt ; Liggend of Staand
```

Als u ReportPrintInfo wilt gebruiken, declareert u een variabele van het ReportPrintInfo-type en wijst u waarden toe aan een of meer velden in het record. U hoeft alleen waarden toe te wijzen aan velden waarin u bent geïnteresseerd. Paradox wijst standaardwaarden toe aan de overige velden. Als u een rapport afdrukt met de ReportPrintInfo-structuur, wordt het dialoogvenster 'Bestand afdrukken' niet geopend. Paradox gaat ervan uit dat u alle benodigde gegevens hebt opgegeven en stuurt het rapport rechtstreeks naar de printer.

Het volgende voorbeeld laat zien hoe u afdrukt met een ReportPrintInfo-record:

```
; afdrukkenMetRecord::pushButton
method pushButton(var eventInfo Event)
   var
     voorraadRap  Report
     rapInfo      ReportPrintInfo
   endVar
   ; bereid eerst rapInfo-record voor
   rapInfo.nCopies = 2
   rapInfo.makeCopies = True
   rapInfo.restartOptions = PrintLock

   ; open het rapport en druk het af met rapInfo
   voorraadRap.open("voorraad.rdl")
   voorraadRap.print(rapInfo)
endmethod
```

Deze code wijst zodanige waarden toe aan de velden van *repInfo* dat twee kopieën van het rapport worden afgedrukt en dat de onderliggende tabellen worden vergrendeld terwijl het rapport wordt afgedrukt.

Raadpleeg verder de online ObjectPAL Help voor meer informatie en voorbeelden.

# TableView: Rijen en kolommen weergeven

Een tabelvenster geeft een tabel in een eigen venster weer. Een tabelvenster verschilt van een tabelframe (een UIObject dat op een formulier is geplaatst) en van een TCursor (een programmeerconstructie die verwijst naar de gegevens in een tabel). Zie het *Handboek* voor meer informatie over het interactieve gebruik van tabelvensters.

Als u een TableView-variabele declareert, maakt u tegelijkertijd een handle voor het tabelvenster. Deze handle is een variabele waarnaar u in uw code kunt verwijzen om het tabelvenster te manipuleren. De volgende instructies declareren bijvoorbeeld de TableView-variabele *orderTV* en gebruiken de variabele vervolgens om de positie van het tabelvenster in te stellen voor de tabel *Order*.

```
var
   orderTV TableView
endVar

if orderTV.open("order.db") then
   orderTV.setPosition(100, 200, 3000, 12000)
else
   msgStop("Stop", "Kan ORDER.DB niet openen")
endIf
```

In deze paragraaf worden de volgende onderwerpen behandeld:

- Methodes van het TableView-type
- Kenmerken van tabelvensters
- Tabelvensters en TCursors

## Methodes van het TableView-type

Methodes van het TableView-type zijn een subgroep van de methodes die voor het Form-type zijn gedefinieerd. U gebruikt deze methodes om het formaat, de positie en de weergave van het tabelvenster te bepalen. U kunt ook de **action**-methode van het TableView-type gebruiken met handelingsconstanten om een tabelvenster te "besturen". De volgende instructies openen bijvoorbeeld een tabelvenster en schuiven door de eerste vijf records:

```
var
   orderTV TableView
   i SmallInt
endVar
if orderTV.open("order.db") then      ; geef een tabelvenster weer
   for i from 1 to 5
      orderTV.action(DataNextRecord)  ; schuif door 5 records
      sleep(500)
```

```
   endFor
else
   msgStop("Stop", "Kan de tabel niet openen.")
endIf
```

### Gegevens bewerken met TableView

U kunt ObjectPAL gebruiken om de gegevens in een tabelvenster te bewerken met een TableView-variabele, zoals het volgende voorbeeld laat zien:

```
var
   orderTV TableView
   i SmallInt
endVar
if orderTV.open("order.db") then     ; geeft een tabelvenster weer
orderTV.action(DataBeginEdit)
   for i from 1 to orderTV.nRecords ; nRecords is een kenmerk, dus geen ()
      if orderTV.Onderdeelnaam = "Tandwiel" then
         orderTV.Onderdeelnaam = "Ding"
      endIf
      orderTV.action(DataNextRecord)
   endFor
   orderTV.action(DataEndEdit)
else
   msgStop("Stop", "Kan de tabel niet openen.")
endIf
```

### wait en tabelvensters

De **wait**-methode van het TableView-type gedraagt zich op een aantal punten anders dan de methode met dezelfde naam voor het Form-type:

- De **wait**-methode van het TableView-type kan geen teruggegeven waarde ontvangen. Omdat u geen code kunt koppelen aan een tabelvenster, is er geen **formReturn**-methode voor het TableView-type. U kunt alleen nog terugkeren uit de **wait**-methode van een TableView door de tabel interactief te sluiten.
- Nadat u de tabel interactief hebt gesloten, moet u nog **close** aanroepen om de tabel van het scherm te verwijderen. Dit is uw laatste kans iets in de tabel te doen voordat deze wordt gesloten. U kunt bijvoorbeeld waarden controleren.

```
method pushButton(var eventInfo Event)
   var
      klantTV TableView
   endVar

   klantTV.open("klant.db")      ; geef de tabel weer
   klantTV.action(DataBeginEdit)
   klantTV.wait()                ; wacht tot de gebruiker de tabel sluit

   if klantTV."Naam".value= "" then ; controleer de waarde van het veld 'Naam'
      msgInfo("Naam", "Voer een naam in.")
      klantTV.bringToTop()
   else
      klantTV.close()
   endIf
endmethod
```

## Kenmerken van tabelvensters

U gebruikt de TableView-variabele om kenmerken van tabelvensters te manipuleren in drie hoofdgebieden:

- Het tabelvenster als geheel (bijvoorbeeld de achtergrondkleur, de rasterstijl, het aantal records en de waarde van het huidige record)
- De gegevens op veldniveau in de tabel (bijvoorbeeld het lettertype, de kleur en de weergave)
- De titelbalk van het tabelvenster (bijvoorbeeld de tekst, het font, de kleur en de uitlijning)

De volgende code opent een tabelvenster, stelt bepaalde kenmerken in, voert verplaatsingen uit in het tabelvenster en geeft de waarden van bepaalde kenmerken weer:

```
var
   klantTV  TableView
   kenWaarde AnyType
endVar

if klantTV.open("klant.db") then

   ; deze opdrachten hebben invloed op de hele tabelweergave

   klantTV.Color = LightBlue                  ; verander kleur van het raster
   klantTV.GridLines.LineStyle = DottedLine   ; verander lijnstijl van raster
   klantTV.CurrentRecordMarker.Show = True    ; geef het huidig recordmerkteken
                                              ; weer
   klantTV.CurrentRecordMarker.Color = Red    ; verander kleur van
                                              ; recordmarkering

   ; deze opdrachten beïnvloeden de gegevens op veldniveau

   kenWaarde = klantTV.NRecords               ; vraag aantal records op,
                                              ; alleen-lezen
   if kenWaarde > 5 then
      klantTV.RowNo= 5                        ; ga naar rijnummer 5
   endIf
   klantTV.fieldNo = 4                        ; ga naar veld 4

   ; deze opdrachten veranderen de titelbalk door het aantal records toe te
voegen aan het huidige bericht

   klantTV.setTitle(KlantTV.GetTitle() + " (" + String(kenWaarde) + " records)")

else
   beep()
   msgInfo("Oeps!", "Kan KLANT.DB niet openen")
endif
```

Raadpleeg Appendix C voor een volledig overzicht van de kenmerken van tabelvensters.

## Tabelvensters en TCursors

U kunt tabelvensters en TCursors op twee manieren met elkaar verbinden:

- U kunt een TCursor met een tabelvenster associëren.

- U kunt een tabelvenster afstemmen op een TCursor.

Deze relaties worden behandeld in de volgende paragrafen.

### TCursor associëren met een tabelvenster

U kunt de **attach**-methode van het TCursor-type gebruiken om een TCursor te associëren met de tabel die wordt weergegeven in een tabelvenster. De volgende instructies declareren bijvoorbeeld de TCursor-variabele *ordTC* en roepen vervolgens **attach** aan om *ordTC* te associëren met de tabel *Order*, die wordt weergegeven in een tabelvenster dat is geopend met de TableView-variabele *ordTV*:

```
var
   ordTC TCursor
   ordTV TableView
endVar

if ordTV.open("order.db") then    ; open een tabelvenster
   ordTC.attach(ordTV)            ; associeer een TCursor met de tabel
else
   msgStop("Stop", "Kan de tabel niet openen.")
endIf
```

Als de TCursor is geassocieerd met het tabelvenster, kunt u TCursor-methodes gebruiken om te werken met de gegevens in de tabel. Dit verbetert vaak de prestaties, omdat de verwerking wordt uitgevoerd in het systeemgeheugen, zonder extra belasting door de weergave.

### Tabelvenster afstemmen op een TCursor

Het TableView-type kent een **moveToRecord**-methode die een tabelvenster afstemt op een TCursor. Met andere woorden: een **moveToRecord**-instructie zorgt ervoor dat een tabelvenster de gegevens in een TCursor weergeeft. De volgende instructies koppelen bijvoorbeeld een TCursor aan een tabelvenster, zoeken een record in de TCursor en stemmen vervolgens het tabelvenster af op de TCursor:

```
var
   klantTC TCursor
   klantTV TableView
endVar

klantTV.open("klant.db")       ; opent een tabelvenster
klantTC.attach(klantTV)        ; associeert een TCursor met het tabelvenster

if klantTC.locate("Naam", "Ouwens") then   ; Zoekt de TCursor voor een waarde.
   klantTV.moveToRecord(klantTC)           ; Als de waarde is gevonden,
else                                       ; stem tabelvenster dan af op TCursor
   msgInfo("Zoeken mislukt.", "Kan Ouwens niet vinden.")
endIf
```

Raadpleeg verder Hoofdstuk 16 voor meer informatie over het gebruik van **attach** en **moveToRecord** met TCursors en UIObjecten.

HOOFDSTUK 15

# Gegevenstypes

Als u tabellen maakt, gebruikt u verschillende veldtypes. Deze vertegenwoordigen verschillende soorten gegevens en de bewerkingen die u wilt verrichten. Dit is met name het geval als u ObjectPAL-applicaties ontwerpt, waarin u verschillende soorten bewerkingen en gegevenstypes nodig hebt om bepaalde taken te verrichten. ObjectPAL ondersteunt de gegevensopmaken die beschikbaar zijn voor uw tabellen en kent speciale gegevenstypes die speciaal zijn bedoeld voor de ontwikkeling van applicaties (zie Tabel 15-1).

Tabel 15-1 Gegevenstypes

| Type | Beschrijving |
|---|---|
| AnyType | Een categorie voor allerlei elementaire gegevenstypes |
| Array | Een geïndexeerde verzameling gegevens |
| Binary | Gegevens die de computer kan lezen |
| Currency | Wordt gebruikt om valutawaarden te manipuleren |
| Date | Kalendergegevens |
| DateTime | Combinaties van kalender- en klokwaarden |
| DynArray | Een dynamische array |
| Graphic | Een bitmap-beeld |
| Logical | True of False |
| LongInt | Wordt gebruikt om grote gehele getallen te vertegenwoordigen |
| Memo | Bevat veel tekst |
| Number | Waarden met zwevend decimaalteken |
| OLE | Een koppeling met een andere applicatie |
| Point | Informatie over een positie op het scherm |
| Record | Een door de gebruiker gedefinieerde structuur |
| SmallInt | Wordt gebruikt om kleine gehele getallen te vertegenwoordigen |
| String | Letters |
| Time | Klokgegevens |

In dit hoofdstuk worden alle gegevenstypes beschreven en wordt uitgelegd hoe de gegevenstypes werken. Daarnaast wordt het gebruik van de gegevenstypes beschreven aan de hand van voorbeelden. (De online ObjectPAL Help geeft een overzicht van de methodes en procedures voor alle types.) Een goed begrip van de werking en het gebruik van de gegevenstypes is een belangrijke voorwaarde om met ObjectPAL te kunnen werken.

# AnyType: een algemeen gegevenstype

AnyType is een gegevenstype waar van alles onder valt. Met dit gegevenstype kunt u methodes voor verschillende gegevenstypes ontwerpen in situaties waarin u het gegevenstype van de doelwaarde pas kent als de methode is uitgevoerd. Een AnyType-waarde kan één van de volgende gegevenstypes zijn:

- Binary
- Currency
- Date
- DateTime
- Graphic
- Logical
- LongInt
- OLE
- Memo
- Number
- Point
- SmallInt
- String
- Time

Een AnyType-waarde kan nooit een complexer type zijn (zoals TCursor of TextStream). Dit gegevenstype erft kenmerken van de waarde die eraan wordt toegewezen. AnyType gedraagt zich als een String als er een String-waarde aan is toegewezen, als een Number als er een Number-waarde aan is toegewezen enzovoort.

## Ongedeclareerde AnyType-variabelen

AnyType-gegevenstypes zijn in ObjectPAL opgenomen om u de mogelijkheid te geven variabelen voor elementaire gegevenstypes te

gebruiken zonder deze eerst te declareren. (Het is wel beter om variabelen te declareren als dit mogelijk is.) Een voorbeeld van het gemak dat Anytype-variabelen bieden: stel dat u een eenvoudige **for...endFor**-structuur wilt maken. U kunt dan de volgende code schrijven, zonder dat u expliciet een gegevenstype voor *x* hoeft te declareren:

```
for x from 1 to 9
   message(x)
endFor
```

Als ObjectPAL een ongedeclareerde variabele tegenkomt, veronderstelt het programma dat de variabele van het type AnyType is en wordt de conversie intern afgehandeld.

## Gedeclareerde AnyType-variabelen

U kunt ook een variabele expliciet als AnyType declareren. Dit is nuttig als u werkt met waarden waarvan het gegevenstype niet bekend is of kan veranderen. Elk ontwerpobject heeft bijvoorbeeld kenmerken (beschreven in Hoofdstuk 13) en de verschillende kenmerken geven waarden van verschillende gegevenstypes terug: het kenmerk 'Name' geeft bijvoorbeeld een String terug en het kenmerk 'Size' geeft een Point terug.

### *Waarom variabelen declareren?*

Code wordt sneller uitgevoerd als u variabelen declareert. Koppel bijvoorbeeld de volgende code aan de ingebouwde **pushButton**-methode van een knop:

```
method pushButton(var eventInfo Event)
   beep()
   for i from 1 to 1500 endFor
   beep()
endmethod
```

Deze code gebruikt de ongedeclareerde variabele *i* als index in de **for...endFor**-lus. Als u het formulier start en op de knop klikt, klinkt er een pieptoon, hoort u enkele seconden niets (hoe lang hangt af van de snelheid van de computer) en klinkt er vervolgens weer een pieptoon.

Voer vervolgens dezelfde code uit, maar nu met de gedeclareerde variabele *i*. Bewerk de **pushButton**-methode van de knop en voeg één regel code toe:

```
method pushButton(var eventInfo Event)
var i SmallInt endVar ; voeg deze regel toe om de variabele i te declareren
   beep()
   for i from 1 to 1500 endFor
   beep()
endmethod
```

Als u nu op de knop klikt, is het interval tussen de pieptonen veel korter, omdat u de variabele *i* hebt gedeclareerd. Paradox hoeft de

variabele niet elke keer dat de lus wordt doorlopen, te testen, zodat de code sneller wordt uitgevoerd.

## AnyType-variabelen met kenmerkwaarden

In het volgende voorbeeld ziet u hoe u AnyType-variabelen gebruikt om met kenmerkwaarden te werken. U maakt een array met de naam *kenmAr*, waarin elk element een kenmerk is. Vervolgens wordt de **getProperty**-methode van het UIObject-type gebruikt om het veldobject met de naam *hetVeld* te inspecteren en de waarde van elk kenmerk in de array toe te wijzen aan de variabele *kenmWaarde*. Omdat *kenmWaarde* een AnyType is, kan deze variabele de kenmerkwaarden opslaan, hoewel elke waarde van een ander gegevenstype is.

```
var
   kenmAr  Array[3] String
   kenmWaarde AnyType
endVar

; sla kenmerknamen op in de array
kenmAr[1] = "name"         ; Het kenmerk 'Name' geeft een String terug
kenmAr[2] = "size"         ; Het kenmerk 'Size' geeft een Point terug
kenmAr[3] = "CursorPos"    : Het kenmerk 'CursorPos' geeft een LongInt terug

for i from 1 to 3
   kenmWaarde=hetVeld.getProperty(kenmAr[i]) ; inspecteer hetVeld, vraag
                                             ; waarden op
   kenmVal.view()                            ; geef de waarde weer in een
                                             ; dialoogvenster
endFor
```

## Operatoren met AnyType-waarden

Als u binaire operatoren gebruikt (operatoren zoals + en -, die slechts op twee waarden tegelijkertijd werken) met AnyType-waarden, probeert ObjectPAL de waarden te converteren naar een gemeenschappelijk gegevenstype. Als u bijvoorbeeld een SmallInt optelt bij een Number, wordt de SmallInt geconverteerd naar een Number voordat de optelling wordt uitgevoerd. Waarden worden altijd geconverteerd naar het meest exacte type. In het geval van een SmallInt en een Number, wordt de SmallInt geconverteerd, omdat Number het meest exacte type is. Een waarde wordt nooit geconverteerd naar een type dat niet alle informatie van het oorspronkelijke type kan bevatten. Een Number wordt bijvoorbeeld nooit geconverteerd naar een SmallInt.

Niet alle types kunnen worden gecombineerd: u kunt bijvoorbeeld geen String optellen bij een Number. De regels die ObjectPAL gebruikt om gegevenstypes automatisch te combineren en te converteren, worden beschreven in Hoofdstuk 11.

## Het kenmerk 'Value' met UIObjecten

UIObjecten (bijvoorbeeld veldobjecten) hebben een kenmerk met de naam *Value*, dat een AnyType oplevert. De volgende instructies

wijzen bijvoorbeeld aan *x* de waarde van het veldobject *eenVeld* toe, ongeacht welk type gegevens *eenVeld* bevat:

```
var x AnyType endVar

x = eenVeld.value
```

Met ObjectPAL kunt u het sleutelwoord *value* in uitdrukkingen weglaten, zodat de volgende twee instructies identiek zijn:

```
x = eenVeld.value

x = eenVeld
```

Deze snelle manier kunt u echter alleen gebruiken als u werkt met een AnyType-waarde of met een waarde die kan worden geconverteerd naar een AnyType-waarde. In de vorige voorbeelden wordt de variabele *x* gedeclareerd als AnyType. Kijk echter eens naar de volgende eigen procedure met de naam *haalObjectSpec*. Deze procedure heeft één argument, een UIObject-variabele met de naam *hetObject*.

```
proc haalObjectSpec(hetObject UIObject)
   msgInfo("Naam", hetObject.name)
   msgInfo("Grootte", hetObject.size)
   msgInfo("Positie", hetObject.position)
endProc
```

Als een instructie *haalObjectSpec* aanroept, geeft deze het UIObject *eenVeld* door en niet de waarde van het veld. Bijvoorbeeld:

```
haalObjectSpec(eenVeld)
```

Hoewel u in sommige gevallen het sleutelwoord *Value* kunt weglaten, is uw code beter te lezen als u het sleutelwoord wel gebruikt.

## Geavanceerd onderwerp: waarden in veldobjecten

Deze paragraaf is bedoeld voor gevorderde programmeurs. Als u een veldobject bewerkt of overschakelt op veldweergave, maakt Paradox een tijdelijk tekstobject boven op het veldobject. Toetsaanslagen, muisklikken en waarden die worden toegewezen door ObjectPAL-instructies verschijnen allemaal eerst in het tijdelijke tekstobject. Als u vervolgens het veldobject verlaat, kopieert Paradox de inhoud van het tekstobject naar het veldobject en wordt het tekstobject verwijderd. In verreweg de meeste gevallen is deze bewerking, zowel voor de gebruiker als voor de ObjectPAL-programmeur, volkomen duidelijk. Houd er echter rekening mee dat het tijdelijke tekstobject weliswaar bestaat, maar dat het gegevenstype van de waarde in het veldobject Memo is. Voor het overige komt het gegevenstype overeen met het natuurlijke type van de gegevens. Als een veldobject bijvoorbeeld een SmallInt-waarde bevat, is het natuurlijke gegevenstype van het veldobject SmallInt.

Als een veldobject verbonden is met een tabel in het gegevensmodel van het formulier, is het gegevenstype van de waarde in het

veldobject hetzelfde als het gegevenstype van het veld in de onderliggende tabel. Als een veld niet-verbonden is, kan het gegevenstype van de waarde in dit veld een van de volgende zijn:

- Als het veldobject leeg is, is het gegevenstype van de waarde String.
- In veldweergave of als u het toetsenbord gebruikt om een waarde in te voeren, is het gegevenstype van de waarde Memo.
- Als u een ObjectPAL-instructie gebruikt om een waarde toe te wijzen, is het gegevenstype van de waarde het natuurlijke type van die waarde. Als u ObjectPAL bijvoorbeeld gebruikt om een String-waarde toe te wijzen aan een veldobject, is String het gegevenstype.
- Als u een waarde uit het Klembord plakt, is het gegevenstype van die waarde het natuurlijke type van de waarde.

In de meeste gevallen is het gegevenstype van de waarde van een veldobject niet van belang, omdat ObjectPAL automatisch waarden tussen de elementaire gegevenstypes converteert. In bepaalde situaties moet u ObjectPAL echter helpen om het gewenste resultaat te bereiken. U zou bijvoorbeeld terecht verwachten dat de volgende instructie **3,1** weergeeft:

```
message(2.1 + 1) ; geeft 3,1 weer
```

Stel echter dat een formulier een niet-verbonden veldobject met de naam *hetVeld* bevat. Als u de waarde 2.1 typt in *hetVeld*, geeft de volgende instructie **3** weer, omdat het gegevenstype van *hetVeld* Memo is (als u het toetsenbord gebruikt om gegevens in een niet-verbonden veld te typen, is het gegevenstype altijd Memo) terwijl het gegevenstype van 1 SmallInt is:

```
message(hetVeld.value + 1) ; geeft 3 weer
```

ObjectPAL converteert de Memo-waarde naar een SmallInt-waarde om de waarden te kunnen optellen. Tijdens deze procedure wordt de waarde rechts van het decimaalteken ingekort.

Er zijn twee manieren om dit verlies van gegevens te voorkomen: de waarde van het veldobject omzetten (casting) naar het gewenste gegevenstype of de gewenste precisie aangeven in de bekende waarde. De volgende instructies geven bijvoorbeeld beide 3,10 weer:

```
message(Number(hetVeld.value) + 1)  ; zet de waarde van het veldobject om naar
                                    ; Number

message(hetVeld.value + 1.00)       ; geef precisie aan in de bekende waarde
                                    ; (gebruik 1.00 in plaats van 1)
```

Beide instructies geven ObjectPAL de informatie die nodig is om het verwachte resultaat te tonen.

# Array: postvakken voor gegevens

Een array bevat waarden (*elementen* genaamd) in *cellen*, vergelijkbaar met de manier waarop een postvak een brief bevat. Een ObjectPAL-array is eendimensionaal, zoals een rij postvakken die elk één element bevatten.

***Opmerking*** In ObjectPAL worden array-waarden geteld vanaf 1 en niet vanaf 0, zoals in sommige andere talen.

Als u arrays in methodes wilt gebruiken, declareert u deze door een naam, een grootte (aantal elementen) en een gegevenstype voor de elementen op te geven.

- Namen. Arrays worden op dezelfde manier benoemd als andere variabelen, volgens de conventies die u vindt in Hoofdstuk 11.
- Grootte. Het maximale aantal elementen dat een array kan bevatten, hangt af van het gegevenstype en het geheugen van het systeem. Arrays kunnen een *statische* grootte (vaste grootte) of een *variabele* grootte (aanpasbare grootte) hebben.

***Opmerking*** ObjectPAL ondersteunt ook dynamische arrays. Zie hiervoor de beschrijving van het DynArray-type in dit hoofdstuk.

- Gegevenstypes. Een array kan gegevens bevatten van elk type, *behalve* Array, DynArray en Record.

## Array declareren

De syntaxis voor de declaratie van een Array luidt:

**var**
  *arrayNaam* **Array**[*arrayGrootte*] *gegevensType*
**endVar**

U geeft een statische array op door de lengte van de array tussen vierkante haakjes te plaatsen ([100] betekent bijvoorbeeld 100 elementen).

U geeft een array met een aanpasbare grootte op met lege vierkante haakjes: [ ]. Voordat u een element aan een array met een aanpasbare grootte kunt toewijzen, moet u **setSize** of **grow** gebruiken om een begingrootte op te geven.

U kunt de volgende declaraties opnemen in een Var-venster of in de sectie voor de declaratie van variabelen van een methode:

```
var
   stringArray Array[300] String   ;  300 String-elementen, statische array
   anyTypeArray Array[300] AnyType ;  300 AnyType-elementen statische array
endVar
```

U kunt ook een array declareren in de sectie voor de declaratie van variabelen van een methode of in het Type-venster van een object. Declaraties van door de gebruiker gedefinieerde types worden als volgt geplaatst in het Type-venster van een ontwerpobject of voorafgaand aan de sectie voor de declaratie van variabelen van een methode:

```
type
   mijnArrayType = Array[100] SmallInt
;  Declareert een Array-type met de naam mijnArrayType.
;  MijnArrayType is statisch (100 SmallInt-elementen).
;  U kunt nu andere arrays declareren van het type mijnArrayType (zie
;  hieronder).

   mijnAndereArrayType = Array[] LongInt
   ; Declareert een ander Array-type: een array met een aanpasbare grootte van
   ; LongInt-elementen.
endType

var
   mijnArray1  mijnArrayType
;  100 SmallInt-elementen, zoals mijnArrayType

   mijnArray2  mijnAndereArrayType
;  variabele array van LongInt-elementen, zoals mijnAndereArrayType
endVar
```

Als u een array wilt gebruiken als eigen methode of procedure, declareert u deze in het declaratievak voor het Type. Zie "Arrays doorgeven als argumenten", verderop in dit hoofdstuk.

## Elementen toewijzen

De toewijzing van elementen aan een array is te vergelijken met het leggen van post in postvakken. In een ObjectPAL-array worden de postvakken cellen genoemd, de inhoud van een cel wordt een element genoemd en elke cel heeft een eigen indexgetal. De eerste cel heeft niet nummer 0, maar nummer 1. De syntaxis voor het toewijzen van elementen aan cellen luidt:

*arrayNaam*[*celNummer*] = *uitdrukking*

Hier volgen enkele voorbeelden:

```
var
   mijnStringArray Array[4] String ; array van 4 Strings
   mijnNumberArray Array[]  Number ; variabele array van Numbers
   n SmallInt
endVar

mijnStringArray[1] = "hallo" ; wijst String hallo toe aan cel 1
mijnStringArray[4] = "tot ziens"
; Plaatst String tot ziens in cel 4. Cel 2 en 3 zijn leeg.

mijnNumberArray.setSize(8) ; maakt cellen voor 8 elementen

mijnNumberArray[1] = 231   ; Wijst Number 231 toe aan element 1

for n from 2 to 8  ; Wijst Number 2 tot en met 8 toe aan cel 2 tot en met 8
   mijnNumberArray[n] = n
endFor
```

De elementen die u toewijst, moeten van een type zijn dat geldig is voor de array. Als u bijvoorbeeld een numerieke waarde toewijst aan een array met reeksen, krijgt u een foutmelding. Als u verschillende gegevenstypes in een array gebruikt, declareert u de array als AnyType. Bijvoorbeeld:

```
var
   mijnArray[] AnyType          ; declareert een variabele array van
                                ; AnyType-elementen
endVar
mijnArray.setSize(2)            ; maakt cellen voor 2 elementen
mijnArray[1] = "kat"            ; slaat een String-waarde op
mijnArray[2] = 5                ; slaat een SmallInt-waarde op
mijnArray.grow(1)               ; voegt een cel toe
mijnArray[3] = Date("11/9/59") ; slaat een Date-waarde op
mijnArray.view()                ; geeft de array weer in een dialoogvenster
```

### *Gegevens toevoegen aan een array met een aanpasbare grootte*

De methodes in Tabel 15-2 voegen gegevens toe aan een array met een aanpasbare grootte. Elke methode breidt de array automatisch uit om plaats te maken voor het element of de elementen die worden toegevoegd.

Tabel 15-2 Methodes om gegevens toe te voegen aan een array met een aanpasbare grootte

| Methode | Beschrijving |
|---|---|
| append | Voegt de inhoud van een array toe aan het einde van een andere array |
| insert | Voegt een of meer lege cellen in een array in |
| addLast | Voegt een element toe na het laatste element van een array |
| insertFirst | Voegt een element in aan het begin van een array |
| insertAfter | Voegt een element in een array in na een opgegeven element |
| insertBefore | Voegt een element in een array in vóór een opgegeven element |
| copyFromArray | Kopieert een record uit een tabel naar een array (of naar een DynArray) |

## Toegang tot elementen in een array

Toegang krijgen tot elementen in een array werkt precies andersom als de toewijzing. De syntaxis luidt:

*expression* = *arrayNaam*[*celNummer*]

Bijvoorbeeld:

```
var
   mijnArray Array[5] SmallInt ; declareer eerst de array

   n, x SmallInt
endVar

for n from 1 to 5
   mijnArray[n] = 2 * n ; wijs waarden toe aan de elementen
endFor
```

```
; vraag toegang tot het eerste en het derde element
x = mijnArray[1] ; geeft 2 terug
x = mijnArray[3] ; geeft 6 terug
```

Erg nuttig is de methode **contains**. Deze methode geeft aan of een array een opgegeven element bevat. Bijvoorbeeld:

```
if deArray.contains("hoi") then
   x = deArray.indexOf("hoi")
else
   msgInfo("deArray", "Element hoi niet gevonden.")
endIf
```

### Gegevens uit een array verwijderen

De volgende drie methodes verwijderen gegevens uit een array met een aanpasbare grootte:

- **remove** verwijdert een of meer elementen uit een array, op basis van het indexgetal.
- **removeItem** verwijdert een waarde als deze de eerste keer voorkomt.
- **removeAllItems** verwijdert een waarde telkens wanneer deze voorkomt.

## Array-operatoren

Met de array-operatoren (+, -, *, /, >, <, >=, <=, <> en =) kunt u rekenkundige bewerkingen op array-waarden uitvoeren. Hier volgen enkele voorbeelden:

```
var SmallInt
   ar1 Array[10] AnyType
   ar2 Array[10] AnyType
   ar3 Array[2] AnyType
   ar4 Array[2] AnyType
   ar5 Array[2] AnyType
endVar
for x from 1 to 10
   ar1[x] = x
endfor
ar2 = ar1 ; kopieert ar1 naar ar2
; Arrays moeten even groot zijn. Als elementen van een verschillend type zijn,
; probeert ObjectPAL het ene type te converteren naar het andere.

ar3[1] = 21
ar3[2] = 15

ar4[1] = 18
ar4[2] = 75

ar5 = ar3 + ar4
ar5.view() ; ar5 = 39, 90
; ar5[1] = ar3[1] + ar4[1] and ar5[2] = ar3[2] + ar4[2]

; uitdrukkingen mogen complex zijn:
ar3 = (ar1 * ar2) + (ar2 / ar1) - ar1
```

U kunt ook array-operatoren gebruiken om arrays te vergelijken. Twee arrays zijn gelijk als deze hetzelfde aantal elementen hebben en

als de elementen voor elke index overeenkomen. Een array is groter (of kleiner) dan een andere array als beide arrays hetzelfde aantal elementen hebben en als elk element in de eerste array groter (of kleiner) is dan het element met het overeenkomende index van de tweede array. Bijvoorbeeld:

```
var ar1, ar2 Array[2] String endVar
ar1[1] = "a"
ar1[2] = "b"

ar2[1] = "a"
ar2[2] = "b"
message(ar2 = ar1) ; Geeft True weer.
                   ; Alle elementen in ar2 = overeenkomende elementen in ar1
sleep(1500)

ar2[1] = "x"
ar2[2] = "z"
message(ar2 > ar1) ; Geeft True weer.
                   ; Alle elementen in ar2 > overeenkomende elementen in ar1

ar2[1] = "a"
message(ar2 > ar1) ; Geeft False weer. ar2[1] = ar1[1]
sleep(1500)

message(ar2 >= ar1) ; Geeft True weer.
                    ; Alle elementen in ar1 >= overeenkomende elementen in ar2
```

De operator <> vergelijkt twee arrays van dezelfde grootte. Deze operator geeft alleen False terug als alle overeenkomende elementen in beide arrays met elkaar overeenstemmen. Als dit niet het geval is, wordt True teruggegeven. Bijvoorbeeld:

```
var
   ar1 Array[2] String
   ar2 Array[2] String
   ar3 Array[3] String
   ar4 Array[2] String
endVar

ar1[1] = "aaa"
ar1[2] = "bbb"

ar2[1] = "aaa"
ar2[2] = "bbb"

ar3[1] = "aaa"
ar3[2] = "bbb"

ar4[1] = "aaa"
ar4[2] = "bb"

message(ar1<>ar2) ; geeft False weer.
sleep(1000)

message(ar1<>ar3) ; Veroorzaakt een runtime fout, omdat
sleep(1000)       ; de arrays niet even groot zijn

message(ar1<>ar4) ; geeft True weer.
sleep(1000)
```

De operator (=) kan zowel worden gebruikt om twee arrays te vergelijken als om de ene array naar de andere te kopiëren. De volgende instructie vergelijkt bijvoorbeeld twee arrays:

```
if ar1 = ar2 then beep() endIf
```

De volgende instructies kopiëren de array *ar1* naar de array *ar2*:

```
var
   ar1 Array[2] String
   ar2 Array[2] String ; U kunt ook een array met een aanpasbare grootte
                       ; gebruiken,
                       ;   d.w.z. ar2 Array[] String
endVar

ar1[1] = "aaa"
ar1[2] = "bbb"

ar2 = ar1  ; kopieer ar1 naar ar2
ar2.view() ; geef ar2 weer in een dialoogvenster.
```

## Arrays doorgeven als argumenten

U kunt een array als argument doorgeven aan een eigen methode of een eigen procedure. U moet dan echter eerst een eigen gegevenstype declareren, zoals u in het volgende voorbeeld ziet.

Alle code in dit voorbeeld is gekoppeld aan de ingebouwde **pushButton**-methode van een knop. Het eerste blok declareert een eigen gegevenstype *GeefAr* (de naam is onbelangrijk). Dit eigen gegevenstype vertegenwoordigt de array die u wilt doorgeven. In dit voorbeeld gaat het om een array van drie tekenreeksen. Het volgende blok declareert een zeer eenvoudige eigen procedure die een argument gebruikt van het type *GeefAr*, met andere woorden, een array van drie reeksen. In het hoofddeel van de methode declareert een **var**-blok een variabele van het type *GeefAr*. Deze variabele wordt doorgegeven aan de eigen procedure **toonAr**. De rest van de code in deze methode initialiseert de array en geeft deze vervolgens door aan **toonAr**. Deze procedure geeft de array vervolgens weer in een **view**-dialoogvenster.

```
type ; declareer eerst een eigen gegevenstype dat de array vertegenwoordigt

array
   GeefAr = Array[3] String
endType

proc toonAr(ar GeefAr) ; deze eigen proc geeft de doorgegeven array weer
   ar.view("U ziet nu...")
endProc

method pushButton(var eventInfo Event)
   var
      ar GeefAr        ; declareer een variabele van het GeefAr-type
   endVar

   ar[1] = "Paradox"   ; initialiseer de array
   ar[2] = "voor"
   ar[3] = "Windows"
```

```
   toonAr(ar)          ; geef de array door aan de eigen proc
endMethod
```

# Binary: gegevens die de computer kan lezen

Een binair object (ook wel BLOB, Binary Large Object, genoemd) bevat gegevens die een computer kan lezen en interpreteren. Een voorbeeld van een binair object is een geluidsbestand: een mens kan het bestand in de ruwe vorm niet lezen of interpreteren, maar een computer wel.

In Tabel 15-3 ziet u de methodes voor het gebruik van Binary-variabelen en binaire objecten.

Tabel 15-3 Methodes van het Binary-type

| Methode | Beschrijving |
|---|---|
| readFromFile | Leest gegevens uit een bestand en slaat deze op in een Binary-variabele |
| size | Geeft het aantal bytes in een binair object |
| writeToFile | Schrijft de inhoud van een Binary-variabele naar een bestand op schijf |

Als u een Binary-variabele declareert, maakt u tegelijkertijd een handle voor een binair object. Een handle is een variabele waarnaar u in uw code kunt verwijzen om binaire gegevens uit te wisselen tussen een bestand op schijf en een tabel, of tussen een bestand op schijf of een tabel en een eigen routine.

De volgende instructies declareren bijvoorbeeld de Binary-variabele *hetGeluid* en lezen binaire gegevens uit een bestand in deze variabele in. Vervolgens wordt de waarde van de variabele toegewezen aan een Binary-veld in een tabel. (Stel dat GELUID.DB een Paradox-tabel is met de volgende structuur: GeluidNaam, A32; GeluidData, B.)

```
var
   geluidTC TCursor
   hetGeluid Binary
endVar
if hetGeluid.readFromFile("herrie.bin") then
   if geluidTC.open("geluid.db") then
      geluidTC.edit()
      geluidTC.insertRecord()
      geluidTC.GeluidNaam = "Herrie"
      geluidTC.GeluidData = hetGeluid
      geluidTC.endEdit()
      geluidTC.close()
   endIf
endIf
```

In het volgende voorbeeld worden binaire gegevens uit GELUID.DB ingelezen in een Binary-variabele, die vervolgens wordt doorgegeven aan een eigen routine.

***Opmerking*** Eigen routines voor de verwerking van binaire gegevens moeten in een andere programmeertaal dan ObjectPAL worden geschreven. De taal moet ook een bestand met de extensie DLL (Dynamic Link Library) kunnen maken. Zie de beschrijving van **uses** in de online ObjectPAL Help voor meer informatie over het gebruik van DLL-bestanden in ObjectPAL. Raadpleeg de documentatie voor de programmeertaal voor meer informatie over het maken van DLL-bestanden.

```
var
   geluidTC TCursor
   hetGeluid Binary
   i Smallint
endVar
if geluidTC.open("geluid.db") then
   for i from 1 to 5
      hetGeluid = geluidTC.GeluidData ; lees binaire gegevens uit het veld
'GeluidData'
      playSound(hetGeluid)           ; geef de gegevens door aan een eigen
                                     ; routine in een DLL
      geluidTC.nextRecord()          ; ga naar het volgende record in de tabel
   endFor
endIf
geluidTC.close()
```

Methodes van het Binary-type worden gedetailleerd beschreven in de online ObjectPAL Help. Raadpleeg ook de beschrijving van het OLE -type verderop in dit hoofdstuk voor de bewerking en de weergave van gegevens met het OLE-protocol.

# Currency: geldwaarden

Currency-waarden kunnen liggen tussen $\pm 3{,}4 * 10^{-4930}$ en $\pm 1{,}1 * 10^{4930}$, met een precisie van zes cijfers achter de komma. De volgende code, verbonden met de **pushButton**-methode van een knop, geeft bijvoorbeeld de waarde 1,1234567 tweemaal weer: eerst als Number-waarde en vervolgens als Currency-waarde. De Currency-waarde wordt afgerond op zes cijfers achter de komma. (In dit voorbeeld wordt verondersteld dat de instellingen voor 'Valutanotatie' van het Configuratiescherm van Windows zeven cijfers achter de komma aangeven.)

```
method pushButton
   var
      nu Number
      cu Currency
   endVar

   nu = 1.1234567
   nu.view()          ; geeft 1,1234567 weer
```

```
   cu = 1.1234567
   cu.view()          ; geeft 1,1234570 weer
endMethod
```

Het aantal weergegeven decimalen hangt af van de manier waarop de gebruiker het Configuratiescherm heeft ingesteld. In een tabel worden echter alle zes de decimalen opgeslagen.

# Date: kalendergegevens

In ObjectPAL kunnen datumwaarden worden weergegeven in de opmaak dag-maand-jaar maand/dag/jaar of dag.maand.jaar.

***Belangrijk*** Data moeten worden omgezet (expliciet gedeclareerd). De volgende code wijst bijvoorbeeld aan *d* de datum 21 december 1997 toe.

```
var d Date endVar
d = Date("12/21/1997")
```

Vergeet de aanhalingstekens rond de datumwaarde niet. Als u de aanhalingstekens weglaat, deelt ObjectPAL de waarden.

De opmaak van datumwaarden wordt bepaald door de instellingen in het Configuratiescherm van Windows of door opmaakinstructies van ObjectPAL.

Hoewel u ObjectPAL kunt gebruiken om berekeningen uit te voeren op elke geldige datum, moeten de datumwaarden die zijn opgeslagen in een Paradox-tabel liggen tussen 1 januari 100 en 31 december 9999.

Een datum in de twintigste eeuw kan worden opgegeven met twee cijfers voor het jaar, bijvoorbeeld:

```
mijnDag = Date("11/09/59")
```

Elke datum tussen de tweede en de tiende eeuw moet drie cijfers voor het jaar bevatten (zoals in 12/17/243). Datums van de elfde tot en met de negentiende eeuw moeten vier cijfers hebben (12/17/1043). Het jaar kan niet in zijn geheel worden weggelaten.

Een datumconstante moet een geldige datum vertegenwoordigen. Paradox weet welke maanden 30 en welke maanden 31 dagen hebben en in welke jaren februari 29 dagen heeft. Paradox kent ook schrikkeleeuwen, als uw datums zo ver reiken. U dient een datum volledig op te geven: u kunt de dag, de maand of het jaar niet weglaten. Met andere woorden, **Date("12/99")** is niet geldig.

U kunt de volgende scheidingstekens in datums gebruiken: tab, spatie, komma (,), punt (.), dubbele punt (:), puntkomma (;), koppelteken (-) of schuine streep (/). U kunt ook het scheidingsteken weglaten.

In Tabel 15-4 ziet u veel gebruikte methodes en procedures die zijn gedefinieerd voor het Date-type.

Tabel 15-4 Methodes van het Date-type

| Naam | Beschrijving |
|---|---|
| day | Haalt de dag uit een Date-waarde |
| dow | Geeft de dag van de week uit een Date-waarde terug |
| month | Haalt de maand uit een Date-waarde |
| moy | Geeft de maand van het jaar uit een Date-waarde terug |
| today | Geeft de huidige datum terug |
| year | Haalt het jaar uit een Date-waarde |

## Berekeningen met datums

Als u berekeningen met Date-waarden uitvoert, is het resultaat een Date-waarde. In het volgende voorbeeld is *Resultaat* een Date-waarde:

```
var D1, D2, Resultaat Date endvar

d1 = Today()
d2 = Date("07/03/92")
Resultaat = D2 - D1    ; Resultaat = 04/15/24
```

Zo kunt u gegevenstypes in complexe bewerkingen combineren, zoals u ziet in het volgende voorbeeld:

```
var
   d1, d2 Date
   t1, t2 Time
   Resultaat DateTime
endvar

d1 = Today()                  ; vraag de huidige datum op
d2 = d1 + 14                  ; telt twee weken op bij de huidige datum
t1 = Now()                    ; vraag de huidige datum op
t2 = t1 + Time("12:00:00")    ; telt twaalf uur op bij de huidige tijd
Resultaat = d2 + t1
Message("Over twee weken en twaalf uur is het ", Resultaat)
```

### *Datumberekeningen omzetten*

Als u berekeningen uitvoert op Date-waarden, kunt u het beste het resultaat omzetten naar andere gegevenstypes om waarden terug te krijgen als het aantal dagen tussen twee datums, zoals in het volgende voorbeeld:

```
var
   d1, d2 Date
   Resultaat SmallInt
endVar

d1 = Today()
d2 = Date("07/21/69")
Resultaat = SmallInt(d1 - d2) ; als het vandaag 3-11-92 is, is dit 8506
Message("De eerste mensen landden ", Resultaat, " dagen geleden op de maan.")
```

Zie Hoofdstuk 11 voor meer informatie over het combineren van gegevenstypes in berekeningen.

# DateTime: kalendergegevens en klokgegevens combineren

Een DateTime-variabele slaat gegevens op in de vorm uur-minuut-seconde-milliseconde jaar-maand-dag. DateTime-waarden worden alleen in ObjectPAL-berekeningen gebruikt. U kunt een DateTime-waarde niet opslaan in een Paradox-tabel. DateTime-waarden moeten worden omgezet (expliciet gedeclareerd). De volgende instructies wijzen bijvoorbeeld aan de DateTime-variabele *dt* een tijd van 10 minuten en 40 seconden over 11 en een datum van 21 december 1997 toe:

```
var dt DateTime endVar
dt = DateTime("11:10:40 am 12/21/97")
```

De waarde moet tussen aanhalingstekens staan.

U kunt de volgende tekens gebruiken als scheidingsteken: tab, spatie, komma (,), koppelteken (-), schuine streep (/), punt (.), dubbele punt (:) en puntkomma (;). U kunt ook het scheidingsteken weglaten.De opmaak van DateTime-waarden wordt bepaald door de instellingen in het Configuratiescherm van Windows of door opmaakinstructies van ObjectPAL.

U dient een DateTime-waarde volledig op te geven. U kunt geen veld weglaten, maar u kunt wel een waarde van 0 voor een veld opgeven.

In Tabel 15-5 ziet u de veel gebruikte methodes die zijn gedefinieerd voor het DateTime-type.

Tabel 15-5 Methodes van het DateTime-type

| Naam | Beschrijving |
|---|---|
| day | Haalt de dag uit een DateTime-waarde |
| dow | Geeft de dag van de week uit een DateTime-waarde terug |
| hour | Haalt de uren uit een DateTime-waarde |
| milliSec | Haalt de milliseconden uit een DateTime-waarde |
| minute | Haalt de minuten uit een DateTime-waarde |
| month | Haalt de maand uit een DateTime-waarde |
| moy | Geeft de maand van het jaar uit een DateTime-waarde terug |
| second | Haalt de seconden uit een DateTime-waarde |
| year | Haalt het jaar uit een DateTime-waarde |

# DynArray: een geïndexeerde lijst

Een DynArray is een flexibel gestructureerde dynamische array. Een dynamische array is een compacte opslagstructuur voor elke combinatie van gegevenstypes. Als u een DynArray gebruikt, kunt u waarden snel opzoeken, zelfs als de dynamische array een groot aantal elementen bevat.

Deze arrays zijn dynamisch omdat u de grootte ervan niet aangeeft. De dimensies van een DynArray veranderen automatisch wanneer er elementen worden toegevoegd of verwijderd. De grootte van een DynArray wordt alleen beperkt door het geheugen van uw systeem.

***Opmerking*** ObjectPAL ondersteunt ook arrays met een vaste grootte en arrays waarvan de grootte kan worden veranderd. Zie de beschrijving van het Array-type voor meer informatie.

In tegenstelling tot arrays met een vaste grootte zijn de indexen van dynamische arrays geen gehele getallen. Indexen van dynamische arrays kunnen bestaan uit elke geldige ObjectPAL-uitdrukking die een String oplevert. Elke index in een dynamische array wordt geassocieerd met een waarde.

## Dynamische arrays declareren

U moet een dynamische array declareren voordat u er waarden in kunt opslaan. Als u een DynArray wilt declareren, geeft u de array een naam en een gegevenstype. Dit kan elk type zijn, *behalve* Array, DynArray of Record. De syntaxis voor de declaratie van een DynArray luidt:

**var**
  *DynArrayNaam* **DynArray[]** *gegevenstype*
**endVar**

*DynArrayName* geeft de naam aan van de DynArray en *dataType* geeft het gegevenstype van de elementen aan. Alle elementen van de array moeten van het gedeclareerde type zijn. U kunt bijvoorbeeld op de volgende manier een dynamische array van reeksen met de naam *reeksDingen* declareren:

```
var
   reeksDingen DynArray[]  String
endVar
```

Als een dynamische array is gedeclareerd, hebben verwijzingen naar elementen de volgende syntaxis:

*DynArrayNaam* [*Label*]=*Waarde*

*DynArrayNaam* moet uniek zijn. *Label* kan elke geldige ObjectPAL-uitdrukking zijn die een reeks oplevert. Het gegevenstype van *Waarde*

moet overeenstemmen met het gegevenstype dat is gedeclareerd voor de DynArray.

## Elementen van dynamische arrays

Als u waarden wilt toewijzen aan de elementen van de dynamische array *Boodschappentas,* kunt u de volgende instructies gebruiken:

```
var
   Boodschappentas DynArray[] AnyType
   s1, s2 String
endVar
Boodschappentas["Type"] = "Papier"
Boodschappentas["Grootte"] = "Groot"
Boodschappentas["Dubbel"] = True
Boodschappentas["Fruit"] = "Appels"
Boodschappentas["Frisdrank"] = "Bronwater"
Boodschappentas["Totaal"] = 3.29
Boodschappentas["Diepvries"] = False
Boodschappentas["Datum"] = Today()
s1 = "Goed"
s2 = "Spul"
Boodschappentas[s1 + s2] = "IJs"
```

Elementen in een dynamische array zijn niet sequentieel geordend (zoals elementen in een vaste array). U vindt de waarde van een element in een dynamische array door te verwijzen naar het *label* (voor de index van het element) in plaats van naar de positie binnen de array.

De volgende instructie geeft bijvoorbeeld de reeks **Groot** weer als de waarde van het element met het label *Grootte*:

```
message(Boodschappentas["Grootte"])
```

Methodes voor het DynArray-type (zoals **contains, size** en **removeItem**) werken hetzelfde als de tegenhangers in het Array-type. Raadpleeg ook de beschrijving van het elementaire taalelement FOREACH in de online ObjectPAL Help, dat opgegeven instructies herhaalt voor elk element in een DynArray.

## DynArray-operatoren

U kunt DynArray-operatoren (= en <>) gebruiken om twee DynArrays te vergelijken en om de ene DynArray naar de andere te kopiëren. Er zijn geen rekenkundige operatoren voor DynArrays.

Twee DynArrays zijn alleen gelijk als alle indexen en alle elementen in de eerste DynArray overeenstemmen met de overeenkomende indexen en elementen in de tweede DynArray; anders zijn de DynArrays niet gelijk. Bijvoorbeeld:

```
var
    da1, da2 DynArray[] String
endVar

da1["Naam"] = "Frank Borland"
da1["Titel"] = "Goeroe"
```

```
da2[""Naam"] = "Frank Borland"
da2["Titel"] = "Genie"

message(da2 = da1) ; geeft False weer
sleep(1500)
message(da2 <> da1) ; geeft True weer
```

U kunt ook de =-operator gebruiken om de ene DynArray naar de andere te kopiëren. Bijvoorbeeld:

```
var
   da1, da2 DynArray[] String
endVar

da1["Naam"] = "Frank Borland"
da1["Titel"] = "Filantroop"

da2 = da1
da2.view() ; da2 is een kopie van da1
```

### copyToArray en copyFromArray

De methodes **copyToArray** en **copyFromArray**, beide gedefinieerd voor de types TCursor en UIObject, verplaatsen gegevens tussen een TCursor en een array of een DynArray. Als u **copyToArray** met een DynArray gebruikt, kopieert **copyToArray** de naam van alle velden in het huidige record van de TCursor met de bijbehorende gegevens. In de zo ontstane DynArray worden de veldnamen als index gebruikt en de gegevens als elementen. Bijvoorbeeld:

```
var
   tc TCursor
   dyn DynArray[] AnyType
endVar

executeQBEFile("haalKlnt.qbe", tc)
tc.copyToArray(dyn)

msgInfo("Naam", dyn["Naam"])

dyn.view()
```

Een belangrijk voordeel van **copyToArray** is dat de DynArray die **copyToArray** maakt, de veldnamen bevat. Een normale array bevat daarentegen alleen de veldwaarden en het vergt dan ook meer werk om de elementen uit een dergelijke array te adresseren.

## Graphic: een beeld

Een Graphic-variabele verschaft een handle om een grafisch object te manipuleren. Met andere woorden, u kunt Graphic-variabelen in ObjectPAL-code gebruiken om grafische objecten te manipuleren. Grafische objecten bevatten en tonen afbeeldingen in de volgende formaten: bitmap (BMP), Encapsulated Postscript (EPS), Graphic Interchange Format (GIF), Paintbrush (PCX) en Tagged Information File Format (TIF).

Met de methodes **readFromClipboard**, **writeToClipboard**, **readFromFile** en **writeToFile** van het Graphic-type kunt u Graphic-variabelen gebruiken om bitmaps over te brengen tussen formulieren (en rapporten), tabellen, het Klembord en bestanden op een schijf.

Stel dat een formulier een tabelframe met de naam *persneelTF* bevat, dat verbonden is met de tabel PERSNEEL.DB. De volgende code leest een grafisch beeld uit een bestand naar een Graphic-variabele en wijst vervolgens de waarde van de variabele toe aan een grafisch veld met de naam *kiekje* in het tabelframe:

```
var
   persneelBMP Graphic ; declareer een Graphic-variabele
endVar

if persneelTF.locate("Naam", "Frank Borland") then
   if persneelBMP.readFromFile("borland.bmp") then
      persneelTF.kiekje.value = persneelBMP
   else
      msgStop("Stop", "Kan het bestand niet lezen.")
   endIf
else
   msgStop("Stop", "Naam niet gevonden.")
endIf
```

## Rasterbewerkingen

De informatie in deze paragraaf heeft geen betrekking op Graphic-variabelen, maar op grafische objecten die in een formulier zijn geplaatst.

Als u een grafisch object interactief definieert, duidt u een *bronafbeelding* aan (het bestand dat u kiest), die moet worden geplaatst in een *bestemming* (het scherm van uw computer). Meestal gaat Paradox ervan uit dat u een onveranderde kopie van het origineel op het scherm wilt plaatsten.

Stel echter dat u interactie wilt tussen de bronafbeelding en het scherm. Misschien wilt u de bronafbeelding transparant maken, zodat de kleur van de pagina erdoorheen schijnt, of wilt u de kleur van de bronafbeelding inverteren. U kunt deze effecten bereiken door Paradox interactief te gebruiken, zoals in het *Handboek* wordt beschreven, en u kunt ObjectPAL gebruiken om de 'RasterOperation'-kenmerken van het grafisch object te manipuleren.

Rasterbewerkingen bepalen hoe Paradox de bronafbeelding combineert met de bestemming: kleuren inverteren, combineren, opnemen, uitsluiten enzovoort. Paradox gebruikt de Booleaanse vergelijkingsoperatoren AND, OR en XOR (exclusieve OR) om afzonderlijke kleurpixels tijdens rasterbewerkingen te combineren.

In Tabel 15-6 wordt beschreven wat de kenmerken van rasterbewerkingen doen.

Tabel 15-6 Gegevenstypes

| 'RasterOperation'-kenmerk | Resultaat op het scherm |
|---|---|
| SourceCopy | Kopieert een onveranderde bronafbeelding naar de bestemming |
| SourcePaint | Gebruikt de Booleaanse operator OR om bronafbeelding en bestemming te combineren |
| SourceAnd | Gebruikt de Booleaanse operator AND om bronafbeelding en bestemming te combineren |
| SourceInvert | Gebruikt de Booleaanse operator XOR om bronafbeelding en bestemming te combineren |
| SourceErase | Inverteert de bestemming en combineert deze met de bronafbeelding met behulp van de Booleaanse operator AND |
| NotSourceCopy | Inverteert de bronafbeelding en kopieert deze naar de bestemming |
| NotSourceErase | Combineert de bronafbeelding met de bestemming en inverteert het resultaat met behulp van de Booleaanse operator OR |
| MergePaint | Inverteert de bronafbeelding en combineert deze met behulp van de Booleaanse operator OR met de bestemming |

U kunt een 'RasterOperation'-kenmerk van een grafisch object instellen door puntnotatie te gebruiken om het object te adresseren of door het object te associëren met een UIObject-variabele. De volgende instructie stelt bijvoorbeeld het kenmerk 'RasterOperation' van het grafische object *visAfbeelding* in op SourcePaint:

```
visAfbeelding.RasterOperation = SourcePaint
```

De volgende instructies declareren een UIObject-variabele met de naam *deAfbeelding*, associëren *deAfbeelding* met een grafisch object met de naam *monaLisa* en stellen het kenmerk 'RasterOperation' van het grafisch object in op SourceInvert.

```
var
   deAfbeelding UIObject
endVar
deAfbeelding.attach(monaLisa)
deAfbeelding.RasterOperation = SourceInvert
```

## Voorbeeldformulier

Het formulier RASTERS.FSL laat zien hoe u het kenmerk 'RasterOperation' gebruikt. Als u de voorbeelden hebt geïnstalleerd toen u Paradox installeerde, kunt u dit formulier starten door uw werk-directory te veranderen in de directory PALVOORB. Kies 'Bestand | Openen | Formulier' en kies vervolgens RASTERS.FSL. Als u de voorbeeldbestanden niet hebt geïnstalleerd, kunt u dit alsnog doen. Volg hiervoor de instructies in *Aan de slag*.

# Logical: True of False

Logische variabelen beantwoorden vaak vragen over andere objecten en bewerkingen, bijvoorbeeld:

- Is deze tabel leeg?
- Wordt dit formulier als pictogram weergegeven?
- Heeft deze bewerking met succes een tekstbestand gemaakt?

Logische variabelen nemen één byte geheugenruimte in beslag. Deze variabelen hebben twee mogelijke waarden: True of False.

## Logische operatoren

De logische operatoren zijn, in volgorde van voorrang, NOT, AND en OR.

```
if NOT mijnTabel.isEmpty() then
   mijnTabel.empty()
endIf

mijnKader.visible = NOT(mijnKader.visible) ; wisselt het kenmerk 'Zichtbaar'

if mijnKader.color = Red OR mijnCirkel.color = Red then
   msgStop("Rood!", "Rood!")
endIf

if x > 5 AND y < 12 OR y > 222 then
; hetzelfde als if (x > 5 AND y < 12) OR (y > 222), omdat AND voorrang heeft op
; OR
   z = 234
else
   z = 1
endIf
```

***Opmerking*** Er is een verschil tussen de vergelijkingsoperator (<>) en de logische operator (NOT). De operator (<>) vergelijkt twee willekeurige gegevenswaarden, en de operator (NOT) keert een logische waarde om.

In ObjectPAL worden uitdrukkingen aan beide zijden van een AND- of OR-instructie bekeken, in tegenstelling tot programmeertalen die korte evaluatie gebruiken.

*Bitsgewijze bewerkingen*

ObjectPAL beschikt ook over methodes die bitsgewijze bewerkingen uitvoeren. De types LongInt en SmallInt bevatten de methodes **bitAND**, **bitOR** en **bitXOR**. Zie de online ObjectPAL Help voor meer informatie.

# LongInt: lange gehele getallen

LongInt-waarden zijn lange gehele getallen. Dit betekent dat deze waarden kunnen worden vertegenwoordigd door een lange reeks cijfers. Een LongInt-variabele neemt vier bytes in beslag.

In ObjectPAL liggen LongInt-waarden in het bereik van -2.147.483.648 tot 2.147.483.647. Als u probeert een waarde buiten dit bereik toe te wijzen aan een LongInt-variabele, ontstaat er een fout. Bijvoorbeeld:

```
var
   x, y, z LongInt
endVar

x = 2147483647 ; de bovenste grenswaarde voor een LongInt-variabele
y = 1
z = x + y ; veroorzaakt een fout
```

Als u wilt werken met grenswaarden, sla het resultaat dan op in een variabele van een geschikt type. Bijvoorbeeld:

```
var
   x, y LongInt
   z    Number   ; declareer z als Number, zodat deze variabele het resultaat
                 ; kan bevatten
endVar

x = 2147483647   ; de bovenste grenswaarde voor een LongInt-variabele
y = 1
z = x + y        ; Deze instructie is OK, omdat z een Number is
                 ; en de grote waarde aan kan
```

***Opmerking*** Methodes uit de runtime bibliotheek die zijn gedefinieerd voor het Number-type werken ook met LongInt-variabelen. De syntaxis is hetzelfde en de teruggegeven waarde is een Number. Deze code werkt bijvoorbeeld, hoewel **sin** is gedefinieerd voor het Number-type:

```
var
   abc LongInt
   xyz Number
endVar
abc = 43
xyz = abc.sin()
```

***Opmerking*** ObjectPAL ondersteunt ook een andere syntaxis:

*methodeNaam* (*objVar*, *argument* [,*argument*])

*methodeNaam* vertegenwoordigt de naam van de methode, *objVar* is de variabele die een object vertegenwoordigt, en *argument* vertegenwoordigt een of meer argumenten. De volgende instructie gebruikt bijvoorbeeld de standaardsyntaxis van ObjectPAL om de sinus van een getal terug te geven:

```
hetGetal.sin()
```

De volgende instructie gebruikt de andere syntaxis:

```
sin(hetGetal)
```

De standaardsyntaxis is het duidelijkst en het meest consistent, maar u kunt de andere syntaxis gebruiken als dit gemakkelijker is.

# Memo: een grote hoeveelheid tekst

Memo-variabelen slaan tekst en opmaakgegevens op in Paradox-tabellen, tot een maximum van 512 MB. Met behulp van een Memo -variabele en het kenmerk 'Value' van een veldobject, kunt u opgemaakte memo's overbrengen tussen formulieren, rapporten en tabellen. U kunt de tekst van een memo, zonder opmaakgegevens, met de methodes **readFromFile** en **writeToFile** van het type Memo uit en naar een bestand op schijf lezen en schrijven. U kunt ook memo's kopiëren naar en plakken uit het Klembord. Hierbij geldt eveneens dat de opmaak verloren gaat.

U kunt ook de =-operator gebruiken om de waarde van een memoveld toe te wijzen aan een Memo-variabele of een String-variabele.

***Opmerking*** Er zijn geen rekenkundige operatoren of vergelijkingsoperatoren voor Memo-variabelen.

Als u de waarde van een memoveld toewijst aan een String-variabele, wordt de memotekst zonder de opmaakgegevens toegewezen. Als u de waarde van een memoveld aan een Memo-variabele toewijst, wordt de tekst met de opmaak toegewezen. Stel dat een tabel met de naam MEMOTEST.DB een opgemaakt memoveld met de naam MemoVeld bevat en dat een formulier een niet-verbonden veldobject met de naam *formVeld* bevat. Als u de volgende methode start, wordt de inhoud van MemoVeld ingelezen in een String-variabele en vervolgens in een Memo-variabele. Daarna wordt de String-variabele weergegeven in *formVeld* (alleen tekst) en vervolgens de Memo-variabele (opgemaakte tekst).

```
var
   s  String
   m  Memo
   tc TCursor
endVar
tc.open("memoTest.db")
s = tc.MemoVeld
m = tc.MemoVeld
formVeld.value = s ; geeft de tekst weer in formVeld
sleep(1500)
formVeld.value = m ; geeft de OPGEMAAKTE tekst weer in formVeld
sleep(1500)
```

In het volgende voorbeeld wordt de inhoud van een tekstbestand ingelezen in een memoveld in een tabel. Stel dat zich in de huidige directory een tabel met de naam *PDAant* bevindt en dat deze tabel de

volgende velden bevat: *ProjDatum*, een Date-veld, en *ProjAant*, een Memo-veld. De **pushButton**-methode voor een knop met de naam *openBestand* opent en bewerkt een nieuw record en voegt dit in de tabel *PDAant* in. Vervolgens wordt de huidige datum in het veld *ProjDatum* geplaatst en wordt de tekst uit het bestand AANTEK.TXT in het veld *ProjAant* geplaatst.

```
; openBestand::pushButton
method pushButton(var eventInfo Event)
var
  MemoBestand Memo
  pTC         TCursor
endVar

if pTC.open("pdAant.db") then     ; open de tabel met projectaantekeningen
  if MemoBestand.readFromFile("aantek.txt") then
    ; als het memobestand met succes is gelezen
    pTC.edit()                    ; bewerk de tabel met projectaantekeningen
    pTC.insertRecord()            ; voeg een nieuw leeg record in
    pTC.ProjDatum = today()       ; vul het veld 'ProjDatum'
    pTC.ProjAant = MemoBestand    ; schrijf memo naar veld 'ProjAant'
    pTC.endEdit()                 ; beëindig bewerkmodus
  endif
  pTC.close()                     ; sluit de TC
endIf
endmethod
```

## Tekst zoeken in memo's

Er zijn geen specifieke methodes voor het zoeken naar tekst in memo's. Wijs daarom de waarde van het memo toe aan een String-variabele en gebruik String-methodes (bijvoorbeeld **advMatch**, **match** en **search**) om naar tekst te zoeken.

Stel dat een formulier is verbonden met de tabel *Wrakken*, die een memoveld bevat met de naam 'Opmerkingen'. In het volgende voorbeeld ziet u hoe u de gebruiker tekst kunt laten typen en vervolgens de tabel kunt laten doorzoeken naar een record waarin het veld 'Opmerkingen' de getypte tekst bevat. Deze code wordt gekoppeld aan de ingebouwde **pushButton**-methode van een knop.

```
var
   memoAlsString, zoekDit String
   nwePos SmallInt
   i, recMarkering LongInt
   schepenTC TCursor
endVar

proc zoekNaarEinde() Logical
   for i from recMarkering to schepenTC.nRecords()
      if zoekOp() = True then
         return True
      endIf
   endFor
   return False
endProc

proc zoekVanafBegin() Logical
   schepenTC.home()
   for i from 1 to recMarkering
      if zoekOp() = True then
```

```
            return True
         endIf
      endFor
      return False
   endProc

   proc zoekOp() Logical
      memoAlsString = schepenTC.Comments
      nwePos = memoAlsString.search(zoekDit)
      if nwePos <> 0 then
         Opmerkingen.moveToRecord(schepenTC)
         Opmerkingen.moveTo()
         Opmerkingen.action(EditEnterMemoView) ; moet in memoweergave zijn om
                                               ; kenmerk
                                               ; 'CursorPos' in te stellen

         Opmerkingen.cursorPos = nwePos - 1    ; Strings beginnen bij 1,
                                               ; veldobjecten beginnen bij 0.
         Opmerkingen.action(SelectRightWord)   ; Selecteer woord rechts van
                                               ; invoegpositie
         return True
      else
         schepenTC.nextRecord()
      endIf
      return False
   endProc

   method pushButton(var eventInfo Event)
      zoekDit = "Typ hier tekst."
      zoekDit.view("Typ tekst om te zoeken:")

      if zoekDit <> "" then
         schepenTC.attach(Opmerkingen)
         recMarkering = schepenTC.recNo()
         switch
            case zoekNaarEinde() : return
            case zoekVanafBegin() : return
            otherwise : msgInfo("Niet gevonden", zoekDit)
         endSwitch
      endIf
   endmethod
```

Variabelen worden eerst gedeclareerd om deze beschikbaar te maken voor de ingebouwde methode en de eigen procedures. De procedure *zoekOp* voert de zoekopdracht uit. De procedure *zoekNaarEinde* geeft aan dat moet worden gezocht vanaf het huidige record tot het eind van de tabel en de procedure *zoekVanafBegin* geeft aan dat moet worden gezocht vanaf het begin van de tabel.

Zie de online ObjectPAL Help voor meer informatie en voorbeelden.

## Number: waarden met een zwevend decimaalteken

Number-variabelen vertegenwoordigen waarden met een zwevend decimaalteken, bestaande uit een signifcant gedeelte (bijvoorbeeld een decimale breuk als 3,224), vermenigvuldigd met een macht van 10. Het siginficante gedeelte kan maximaal 18 significante cijfers bevatten en de macht van 10 kan liggen tussen $\pm 3{,}4 * 10^{-4930}$ en $\pm 1{,}1 * 10^{4930}$.

Een poging om een waarde buiten dit bereik toe te wijzen aan een Number-variabele, veroorzaakt een fout.

**Opmerking** Methodes uit de runtime bibliotheek en de procedures die zijn gedefinieerd voor het Number-type werken ook met de variabelen LongInt en SmallInt. De syntaxis is hetzelfde en de teruggegeven waarde is een Number. De volgende code werkt bijvoorbeeld, hoewel **sin** niet verschijnt in de lijst van methodes voor het LongInt-type:

```
var
   abc LongInt
   xyz Number
endVar
abc = 43
xyz = abc.sin()
```

**Opmerking** ObjectPAL ondersteunt ook een andere syntaxis:

*methodeNaam (objVar, argument [,argument])*

*methodeNaam* vertegenwoordigt de naam van de methode, *objVar* is de variabele die een object vertegenwoordigt, en *argument* vertegenwoordigt een of meer argumenten. De volgende instructie gebruikt bijvoorbeeld de standaardsyntaxis van ObjectPAL om de sinus van een getal terug te geven:

```
hetGetal.sin()
```

De volgende instructie gebruikt de omgekeerde syntaxis:

```
sin(hetGetal)
```

De standaardsyntaxis is het duidelijkst en het meest consistent, maar u kunt de andere syntaxis gebruiken als dit gemakkelijker is.

**Opmerking** De opmaak van numerieke methodes kan verschillen, afhankelijk van de getalopmaak van Windows op het systeem van de gebruiker, maar de interne weergave van ObjectPAL is altijd hetzelfde.

## Numerieke constanten

Numerieke constanten worden geschreven als een cijferreeks, eventueel voorafgegaan door een min-teken (–) voor negatieve getallen, en optioneel gescheiden door een decimaalteken.

U kunt getallen letterlijk typen, maar u kunt ook de wetenschappelijke notatie gebruiken om getallen te vertegenwoordigen. Dit is vooral gemakkelijk voor zeer grote en zeer kleine getallen. De wetenschappelijke notatie begint met de decimale waarde en eindigt met de letter *E*, gevolgd door de exponent, die positief (+) of negatief (–) kan zijn.

In Tabel 15-7 worden enkele voorbeelden gegeven van numerieke constanten:

Tabel 15-7 Numerieke constanten

| Constante | Beschrijving |
|---|---|
| 25 | Het getal 25 |
| 3.1514 | Een getal met een decimaalteken |
| -17.00000001 | Een negatieve waarde met een kleine decimale waarde |
| 5.6E+9 | $5{,}6 * 10^9$, of 5.600.000.000 |

Numerieke constanten kunnen geen dollartekens of komma's bevatten. Als u numerieke constanten tussen haakjes plaatst, worden deze daardoor niet negatief.

Het numerieke type geeft een getal op met een precisie van achttien cijfers achter de komma. Numerieke constanten kunnen worden gebruikt voor Number-, Currency-, SmallInt- en LongInt-waarden. De grenswaarden zijn afhankelijk van het gegevenstype.

# OLE: Object Linking and Embedding

OLE is de afkorting van Object Linking and Embedding, een protocol dat toegang verschaft tot de functies van een andere applicatie zonder dat u Paradox hoeft te verlaten en zonder dat u die applicatie hoeft te openen als u iets wilt veranderen.

Stel dat een tabel bitmap-afbeeldingen bevat en dat u een Paradox-applicatie wilt maken waarmee de gebruiker deze afbeeldingen kan bewerken. U kunt de afbeeldingen dan maken met een teken-programma dat een OLE-server is (dit wordt hieronder uitgelegd) en vervolgens kunt u de ObjectPAL-methodes van het OLE-type gebruiken om de functies van het tekenprogramma beschikbaar te maken voor de gebruiker (er uiteraard van uitgaand dat de gebruiker het tekenprogramma heeft geïnstalleerd).

***Opmerking*** ObjectPAL en Paradox ondersteunen ook DDE (Dynamic Data Exchange). Dit is ook een protocol voor de uitwisseling van gegevens. Het DDE-type wordt samen met de systeemgegevensobjecten besproken in Hoofdstuk 17, omdat DDE-gegevens niet in een tabel kunnen worden opgeslagen.

In Tabel 15-8 vindt u definities van termen die vaak terugkomen bij de bespreking van OLE-bewerkingen:

Tabel 15-8 OLE-termen en definities

| Termen | Definitie |
|---|---|
| OLE-server | Een applicatie die via het OLE-mechanisme toegang kan verschaffen tot de eigen documenten. Paradox is geen OLE-server. |
| OLE-client | Een applicatie die het OLE-mechanisme kan gebruiken om toegang te krijgen tot documenten die zijn gemaakt door een OLE-server. Paradox is een OLE-client. |
| OLE-object | Een document dat is gemaakt met een OLE-server. Het document bevat de gegevens die u wilt gebruiken in de Paradox-applicatie. |
| OLE-variabele | Een ObjectPAL-variabele die is gedeclareerd als OLE-type. Een OLE-variabele biedt een handle voor de manipulatie van een OLE-object. Met andere woorden, u kunt OLE-variabelen in ObjectPAL-code gebruiken om OLE-objecten te manipuleren. |

Tabel 15-9 geeft een overzicht van de methodes voor het werken met OLE-objecten en OLE-variabelen. Met deze methodes kunt u OLE-objecten lezen, schrijven en gebruiken.

Tabel 15-9 Methodes van het OLE-type

| **Methode** | **Beschrijving** |
|---|---|
| canReadFromClipboard | Geeft True terug als het OLE-object vanaf het Klembord kan worden ingelezen in een OLE-variabele; anders wordt False teruggegeven. |
| edit | Start de server en laat de gebruiker het object bewerken of een andere handeling verrichten. Paradox wacht niet tot de server is gesloten. ObjectPAL wordt zonder onderbreking verder uitgevoerd. |
| enumVerbs | Vult een DynArray met handelingsopdrachten (*werkwoorden* genaamd). De index van de DynArray is een reeks die de naam van het werkwoord vertegenwoordigt. De elementen zijn gehele getallen die de te verrichten handelingen vertegenwoordigen. |
| getServerName | Geeft een reeks terug die de server aanduidt. De reeks is een beschrijvende naam die niet hetzelfde hoeft te zijn als de bestandsnaam van de server. |
| readFromClipboard | Leest (plakt) een OLE-object vanaf het Klembord in een OLE-variabele. Dit lukt niet als het Klembord geen OLE-object bevat. Als u **readFromClipboard** gebruikt, hebben veranderingen die in het OLE-object zijn gemaakt geen invloed op het onderliggend bestand. |
| writeToClipboard | Schrijft de inhoud van een OLE-variabele als een OLE-object naar het Klembord. Deze methode kopieert het oorspronkelijke object alsof de server de kopie heeft gemaakt omdat Paradox niet een OLE-server is. |

## Overzicht van OLE

ObjectPAL kan een OLE-object op twee manieren opvragen:

- Vanuit een tabel
- Vanaf het Klembord

### OLE vanuit een tabel

Als u een OLE-object vanuit een tabel wilt opvragen, declareert u een OLE-variabele. Bijvoorbeeld:

```
var
   oleVar OLE
endVar
```

Wijs vervolgens aan de OLE-variabele de waarde van een OLE-veld toe. Bijvoorbeeld:

```
oleVar = oleTabel.oleVeld.value
```

Roep **edit** aan. De methode **edit** gebruikt twee argumenten: een reeks en een geheel getal. Paradox geeft de reeks door aan de OLE-server, die met de reeks de gebruiker kan informeren over wat er gebeurt. Paradox geeft het gehele getal aan de server door om aan te geven welke handeling moet worden verricht. Paradox wacht niet tot de server is gesloten. ObjectPAL wordt zonder onderbreking verder uitgevoerd.

*Voorbeeld: OLE gebruiken met de Geluidsrecorder*

Het volgende voorbeeld gebruikt een OLE-variabele om geluidsgegevens uit een tabel op te vragen. De code is gekoppeld aan de **pushButton**-methode van een knop en gaat ervan uit dat er een tabel met de naam GELUID.DB bestaat en dat deze tabel twee velden bevat: een alfanumeriek veld met de naam GeluidNaam en een OLE-veld met de naam GeluidData. Als **enumVerbs** wordt aangeroepen, wordt de DynArray *oleWerkw* gevuld met de werkwoorden (handelingen) die worden ondersteund door de OLE-server (Geluidsrecorder). Vervolgens wordt de lijst getoond in een pop-up menu. U kunt een element uit het pop-up menu kiezen om aan te geven welke handeling u wilt verrichten.

```
var oleVar OLE endVar  ; Declareer OLE-variable buiten hoofddeel methode
method pushButton(var eventInfo Event)
   var
      oleTC TCursor
      oleWerkw DynArray[] SmallInt
      werkwPop PopUpMenu
      werkwKeuze, i String
      gekozenWerkw SmallInt
   endvar

   oleTC.open("geluid.db")
   oleVar = oleTC.GeluidData

   oleVar.enumVerbs(oleWerkw)
   forEach i in oleWerkw
      werkwPop.addText(i)
   endForEach
```

```
    werkwKeuze = werkwPop.show()
    gekozenWerkw = oleWerkw[werkwKeuze]
    if not gekozenWerkw.isBlank() then
       oleVar.edit("PdoxWin", gekozenWerkw)   ;Paradox wacht niet op de server
                                              ;ObjectPAL-code wordt verder
                                              ; uitgevoerd
    endIf
    mesgInfo ("Opmerking", "ObjectPAL-code wordt verder uitgevoerd.")
endmethod
```

### OLE vanaf het Klembord

U gaat als volgt te werk om een OLE-object te plaatsen vanaf het Klembord:

1. Open de applicatie van de OLE-server en gebruik deze om een document te openen of te maken. Dit document is het OLE-object.
2. Gebruik de OLE-server om het OLE-object naar het Klembord te kopiëren.
3. Roep **canReadFromClipboard** aan in ObjectPAL om er zeker van te zijn dat de bewerking mogelijk is. Deze stap is optioneel.
4. Roep **readFromClipboard** aan om het OLE-object op te vragen vanaf het Klembord en het toe te wijzen aan een OLE-variabele.
5. Roep **edit** aan.
6. Roep desgewenst **writeToClipboard** aan om het OLE-object naar het Klembord te kopiëren, zodat het beschikbaar is voor een andere OLE-client.

*Voorbeeld: OLE met Paintbrush*

Het volgende voorbeeld gebruikt OLE om een afbeelding te bewerken in Paradox.

1. Open de Paintbrush-applicatie die bij Windows wordt geleverd, en gebruik deze applicatie om een afbeelding te openen of te maken.

**Opmerking**

In het *Handboek* wordt uitgelegd hoe u OLE en Paradox interactief gebruikt.

2. Gebruik Paintbrush om de afbeelding naar het Klembord te kopiëren. (Raadpleeg uw Windows-documentatie voor informatie over het gebruik van Paintbrush.)

**Opmerking**

Als de OLE-server Dynamic Data Exchange (DDE) ondersteunt en u de DDE-opdrachten van de applicatie voor de aansturing van de OLE-server kent, kunt u deze stappen automatiseren. Raadpleeg de documentatie van de applicatie.

3. Maak in Paradox een OLE-object en twee knoppen. Noem de knoppen *bewerkKnop* en *bijwerkKnop*. Noem het OLE-object *oleObject* en koppel de volgende code aan de ingebouwde **pushButton**-methode van de knop:

*pagina::Var-venster*

```
var oleAfbeelding OLE endVar
```

*bewerkKnop::pushButton*

```
method pushButton(var eventInfo Event)
   var
      oleWerkw DynArray[] SmallInt
      werkwPop PopUpMenu
      gekozenWerkw SmallInt
      werkwKeuze String
   endVar

    if oleAfbeelding.canReadFromClipboard() then
      oleAfbeelding.readFromClipboard()
      oleVar.enumVerbs(oleWerkw)
      forEach i in oleWerkw
         werkwPop.addText(i)
      endForEach

      werkwKeuze = werkwPop.show()
      gekozenWerkw = oleWerkw[werkwKeuze]
      if not gekozenWerkw.isBlank() then
         oleVar.edit("PdoxWin", gekozenWerkw)
      endIf

   else
      msgStop("Stop", "Kan de OLE-afbeelding niet lezen vanaf Klembord.")
   endIf

   oleObject.value = oleAfbeelding
   oleObject.endEdit()
endMethod
```

*bijwerkKnop::pushButton*

```
method pushButton(var eventInfo Event)
   oleObject.edit()
   oleObject.value = oleAfbeelding
   oleObject.endEdit()
endmethod
```

4. Start het formulier en klik op *bewerkKnop*. Paintbrush wordt geopend en als het goed is, kunt u de afbeelding bewerken. Als er een probleem is, verschijnt er een dialoogvenster. Sluit Paintbrush af en keer terug naar Paradox. Klik vervolgens op *bijwerkKnop* om *oleObject* bij te werken met de bewerkte afbeelding.

Zie de online ObjectPAL Help voor meer informatie.

# Point: een positie op het scherm

Een Point-variabele bevat informatie over een punt op het scherm. Voor ObjectPAL is het scherm een twee-dimensionaal raster met de oorsprong in de linkerbovenhoek van het insluitend object van het ontwerpobject. De positieve x-waarden lopen naar rechts op en de positieve y-waarden lopen naar beneden op. Een Point heeft een x-waarde en een y-waarde, waarbij $x$ en $y$ worden gemeten in twips (een twip is 1/1440 inch, 1/20 van een printerpunt.)

Point-methodes vragen en geven informatie over schermcoördinaten en relatieve posities van punten. De grootte- en de positiekenmerken van een ontwerpobject worden bijvoorbeeld opgegeven in punten. Open het voorbeeldformulier *Paradox Teken* in de directory waarin u de voorbeeldbestanden van ObjectPAL hebt geïnstalleerd (normaal gesproken C:\PDOXWIN\PALVOORB) als u wilt zien hoe u Point-waarden kunt gebruiken.

***Opmerking*** ObjectPAL berekent de puntwaarden van een ontwerpobject ten opzichte van het insluitend object van dit object. Als een kader bijvoorbeeld een knop bevat, berekent ObjectPAL de relatieve positie van de knop ten opzichte van het kader. Als de knop zich op een lege pagina bevindt, berekent ObjectPAL de relatieve positie ten opzichte van de pagina. Methodes die Point-waarden opvragen of als argument teruggeven, werken ook binnen dit kader.

## Point-operatoren

U kunt Point-operatoren gebruiken (+, -, =, <, >, <=, en >=) om Point-variabelen op te tellen, af te trekken en te vergelijken. Deze operatoren werken op de x-coördinaten van een punt en daarna op de y-coördinaten. Bijvoorbeeld:

```
var
   p1, p2, p3 Point
endVar

p1 = Point(10, 30)
p2 = Point(10, 30)
p3 = Point(10, 33)

message(p1 + p2)  ; Geeft (20, 60) weer, want 10 + 10 = 20, en 30 + 30 = 60.
message(p1 = p2)  ; Geeft True weer. x- en y-coördinaten zijn gelijk.
message(p1 = p3)  ; Geeft False weer. Beide coördinaten moeten gelijk zijn.
message(p3 > p1)  ; Geeft False weer. Beide coördinaten moeten groter zijn.
message(p3 >= p1) ; Geeft True weer. Beide coördinaten zijn groter of gelijk.
```

# Record: een door de gebruiker gedefinieerd gegevenstype

ObjectPAL beschikt kent het type Record, een programmeerbare, door de gebruiker gedefinieerde verzameling informatie. Een Record is te vergelijken met een **record** in Pascal of een **struct** in C.

***Belangrijk*** Records die zijn gedefinieerd als ObjectPAL-gegevenstype, staan los van de records die zijn geassocieerd met een tabel.

De syntaxis voor de declaratie van een Record-gegevenstype luidt:

**type**
***RecordNaam* = Record**
***veldNaam veldType***
**[*veldNaam veldType*]**
*****

**endRecord**
**endType**

Hier staan een of meer *veldNamen* voor de velden (kolommen) van het record en *veldType* is een van de elementaire gegevenstypes. U declareert records in het Type-venster van een ontwerpobject (zie Hoofdstuk 11 voor meer informatie). De volgende code declareert bijvoorbeeld een gegevenstype met de naam *OndrdeelRec* als Record-type. Het record *OndrdeelRec* heeft drie velden: *OndrdeelNaam*, *OndrdeelGetal* en *hoeveelheid*.

```
type
OndrdeelRec = record
                  ondrdeelNaam     String
                  ondrdeelGetal    String
                  hoeveelheid  SmallInt
          endRecord
endType
```

Als u het type hebt gedeclareerd, kunt u andere variabelen als volgt declareren voor het type *OndrdeelRec*:

```
var
   mijnOndrdeelRec OndrdeelRec
endVar
```

Vervolgens wijst u als volgt waarden toe aan de velden van het record:

```
mijnOndrdeelRec.ondrdeelNaam = "tandwiel"
mijnOndrdeelRec.ondrdeelGetal = "WW120A"
mijnOndrdeelRec.hoeveelheid = 174
```

U kunt **view** gebruiken om de waarden van het Record in een dialoogvenster weer te geven. Bijvoorbeeld:

```
mijnOndrdeelRec.view()
```

## Record-operatoren

U kunt Record-operatoren (= en <>) gebruiken om Record-variabelen te vergelijken en toe te wijzen. De records moeten van hetzelfde type zijn. Bijvoorbeeld:

```
type
OndrdeelRec = Record
                  ondrdeelNaam     String
                  ondrdeelGetal    String
                  hoeveelheid  SmallInt
          endRecord
endType

var
   recEen, recTwee OndrdeelRec
endVar

recEen.ondrdeelNaam = "tandwiel"
recEen.ondrdeelGetal = "WW120A"
recEen.hoeveelheid = 174

recTwee = recEen ; wijs waarden van recEen toe aan recTwee (kopiëren)
```

```
recTwee.ondrdeelNaam = "ding" ; wijs nieuwe waarde toe aan veld 'ondrdeelNaam'
                              ; van recTwee

if recEen <> recTwee then
   beep()   ; laat pieptoon horen, omdat records niet hetzelfde zijn
endIf
```

# SmallInt: kleine gehele getallen

SmallInt-waarden zijn kleine gehele getallen. Dit betekent dat deze waarden kunnen worden vertegenwoordigd door een kleine (korte) reeks cijfers. Een SmallInt-variabele neemt twee bytes geheugenruimte in beslag.

SmallInt-waarden liggen in ObjectPAL tussen -32.768 en 32.767. Een poging om een waarde buiten dit bereik toe te wijzen aan een SmallInt-variabele, veroorzaakt een fout. Bijvoorbeeld:

```
var
   x, y, z SmallInt
endVar

x = 32767 ; de bovenste grenswaarde voor een SmallInt-variabele
y = 1
z = x + y ; veroorzaakt een fout
```

Als u met grenswaarden wilt werken, sla het resultaat dan op in een variabele van een type dat dit resultaat ook kan bevatten. Bijvoorbeeld:

```
var
   x, y SmallInt
   z    LongInt   ; declareer z als LongInt, zodat deze variabele het resultaat
                  ; kan bevatten
endVar

x = 32767 ; de bovenste grenswaarde voor een SmallInt-variabele
y = 1
z = x + y

z.view() ; geeft 32768 weer, omdat z een LongInt is
         ; en de grote waarde aan kan
```

***Opmerking*** De waarde -32.768 kan niet worden opgeslagen in een Paradox-tabel, want -32.768 = Leeg volgens Paradox. U kunt deze waarde echter wel gebruiken in berekeningen en u kunt de waarde in een dBASE-tabel opslaan.

***Opmerking*** Methodes uit de runtime bibliotheek en procedures die zijn gedefinieerd voor het Number-type werken ook met LongInt- en SmallInt-variabelen. De syntaxis is hetzelfde en de teruggegeven waarde is een Number. De volgende code wordt bijvoorbeeld uitgevoerd, hoewel **sin** niet verschijnt in de lijst van methodes voor het SmallInt-type:

```
var
   abc LongInt
   xyz Number
endVar
abc = 43
xyz = abc.sin()
```

***Opmerking*** ObjectPAL ondersteunt ook een andere syntaxis:

*methodeNaam (objVar, argument [,argument])*

Hier vertegenwoordigt *methodeNaam* de naam van de methode, *objVar* is de variabele die een object vertegenwoordigt, en *argument* vertegenwoordigt een of meer argumenten. De volgende instructie gebruikt bijvoorbeeld de standaardsyntaxis van ObjectPAL om de sinus van een getal terug te geven:

```
hetGetal.sin()
```

De volgende instructie gebruikt de andere syntaxis:

```
sin(hetGetal)
```

De standaardsyntaxis is het duidelijkst en het meest consistent, maar u kunt de andere syntaxis gebruiken als dit gemakkelijker is.

# String: een tekenreeks

Een String-variabele kan maximaal 32.767 tekens bevatten (gebruik Memo-variabelen voor langere tekst). U gebruikt dubbele aanhalingstekens (" ") om een lege reeks te vertegenwoordigen. String-variabelen nemen per teken één byte geheugenruimte in beslag.

## Reeksen tussen aanhalingstekens

Plaats tekenreeksen in methodes tussen dubbele aanhalingstekens (" "). Doe dit als u gegevens in een tabel typt of bewerkt, als u vaste tekst typt in een rapport- of formulierspecificatie, als u een veld opgeeft, en als u kenmerken van ontwerpobjecten instelt.

***Opmerking*** Een reeks tussen aanhalingstekens kan maximaal 255 tekens bevatten, maar een String-variabele kan maximaal 32.767 tekens bevatten. Bijvoorbeeld:

```
var
   a, b, c, d, e, f, g String
endVar

; deze reeksen tussen aanhalingstekens bevatten 50 tekens
a = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
b = "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb"
c = "ccccccccccccccccccccccccccccccccccccccccccccccccc"
d = "ddddddddddddddddddddddddddddddddddddddddddddddddd"
e = "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
f = "fffffffffffffffffffffffffffffffffffffffffffffffff"

g = a + b + c + d + e + f
```

```
g.view()
; g kan meer dan 255 tekens bevatten, omdat
; het een String-variabele is en geen reeks tussen aanhalingstekens.
```

Reeksen tussen aanhalingstekens worden opgeslagen in een resource-bestand van Windows (raadpleeg uw Windows-documentatie voor informatie) en kunnen worden bewerkt met behulp van de Resource Workshop van Borland. U hoeft Paradox dus niet te starten om deze reeksen te bewerken.

***Opmerking*** De methodes **fill**, **pattern** en **space** van het String-type kunnen niet werken met reeksen die langer zijn dan 1.000 tekens.

## Werken met reeksen

U kunt String-methodes gebruiken om reeksen te combineren en te vergelijken, om reeksen te zoeken en te vervangen en om hoofdletters in kleine letters te veranderen en andersom.

### *Reeksen combineren*

U kunt reeksen combineren met de +-operator. Dit wordt aaneenschakeling genoemd. Bijvoorbeeld:

```
mijnNaam = "Frank " + "Borland" ; slaat "Frank Borland" op in mijnNaam
message("Dhr. " + mijnNaam) ; geeft Dhr. Frank Borland weer
```

### *Reeksen vergelijken*

U kunt ook de vergelijkingsoperatoren =, <>, <, >, <=, en >= gebruiken met reeksen. De =- operator en de <>-operator kijken beide naar exact overeenstemmende waarden, maar u kunt **ignoreCaseInStringCompares** gebruiken, zodat bewerking geen onderscheid maken tussen hoofdletters en kleine letters. De operatoren < en > vergelijken de tekens in de reeksen één voor één. Als tekens gelijk zijn, wordt het volgende teken gecontroleerd. Dit gaat door totdat tekens worden gevonden die niet overeenstemmen.

```
var
   bb String
endVar

bb = "Doet-ie het of doet-ie het niet,"
bb = bb + " dat is de vraag." ; aaneenschakeling
; resultaat is "Doet-ie het of doet-ie het niet, dat is de vraag."

if bb = "hallo" then            ; vergelijking
   bericht("Ze stemmen overeen!")
else
   bericht("Geen overeenstemming.")
endIf         ; geen overeenstemming

message("A" < "B") ; geeft True weer
sleep(2000)
message("a" < "B") ; geeft False weer
sleep(2000)

message("A" < "a") ; geeft True weer
sleep(2000)
message("A" = "a") ; geeft False weer
sleep(2000)
ignoreCaseInStringCompares(True)
```

```
message("A" = "a") ; geeft True weer
sleep(2000)

message("Paradocks" < "Paradox") ; geeft True weer
; bij het eerste teken dat niet overeenstemt, "c" < "x"
```

### Zoeken

U gebruikt **advMatch** of **match** om in een bepaalde reeks naar een andere reeks te zoeken. Beide methodes zijn krachtig en kennen veel opties. Als u een bepaald deel van een reeks wilt teruggeven, gebruikt u **substr**. Zie "String" in de online ObjectPAL Help voor meer informatie en voorbeelden.

### Hoofdletters omzetten in kleine letters en andersom

Als u een reeks wilt omzetten van hoofdletters naar kleine letters, gebruikt u **lower**. Als u een reeks wilt omzetten in hoofdletters, gebruikt u **upper**.

```
var s1 endVar
s1 = "abcd"
message(s1.upper()) ; geeft ABCD weer
sleep(2000)
message(s1.lower()) ; geeft abcd weer
sleep(2000)
```

### Lege reeksen

Een lege reeks wordt aangegeven met twee dubbele aanhalingstekens (" "), *niet* met een spatie (ANSI 32).

### Andere voorbeelden

Hier volgen nog enkele voorbeelden van geldige reeksen:

```
mijnTabel."Nettowinst".font.color = "Green"
; 'Nettowinst' is een veld uit mijnTabel
; "Green" is de kleur van het veld 'Nettowinst'
msgInfo(" ", "Hallo, wereld") ; geeft "Hallo, wereld" weer in een dialoogvenster
msgInfo(" ", "Deze reeks
beslaat twee regels.")        ; geeft een reeks van twee regels weer in een
                              ; dialoogvenster
```

### Backslash-codes in reeksen

Voor bepaalde veel gebruikte ANSI-codes, zoals het tab-teken (ANSI 9) worden speciale backslash-reeksen gebruikt, die vergelijkbaar zijn met de codes in de programmeertaal C. Deze codes, die u vindt in Tabel 15-10, krijgen in reeksen de voorkeur, omdat deze beter leesbaar zijn dan de numerieke equivalenten.

Gebruik bijvoorbeeld een dubbele backslash in een reeks tussen aanhalingstekens voor een directorypad:

```
order.open("c:\\pdoxwin\\formlrn\\order.fsl")
```

Tabel 15-10 Backslash-codes

| Teken | Functie |
|---|---|
| \a | Bel (^G) |
| \b | Backspace |
| \f | Papierdoorvoer(^L) |
| \n | Nieuwe regel (^J) |
| \r | Return (^M) |
| \t | Tab (^I) |
| \v | Verticale tab |
| \" | Dubbel aanhalingsteken (") |
| \\ | Backslash (\) |
| \xxx | ASCII-code van drie cijfers |

Tekens die niet voorkomen in Tabel 15-10 kunnen niet worden voorafgegaan door een backslash. Dit zou een syntaxisfout veroorzaken als u de code compileert.

## Format en uitvoer

U kunt **format** gebruiken om te bepalen hoe een reeks wordt uitgevoerd. Met **format** kunt u het volgende doen:

- Numerieke precisie bepalen
- Gegevens uitvullen en uitlijnen
- Hoofdletters veranderen in kleine letters en andersom
- De vorm bepalen waarin datums worden weergegeven
- Voorloopspaties aan het begin van waarden onderdrukken

U kunt ook verschillende opmaakkenmerken combineren door de specificatie met komma's te scheiden. "W6,AL" geeft bijvoorbeeld aan dat er links wordt uitgelijnd binnen een breedte van zes. U kunt ook opmaakkenmerken combineren achter één *E*. De specificatie "E$C" geeft bijvoorbeeld een zwevend dollarteken en duizendscheiders (na iedere drie cijfers) op.

In de volgende paragrafen ziet u enkele voorbeelden van het gebruik van **format**. Complete specificaties vindt u bij het onderwerp **format** in "String" in de online ObjectPAL Help.

***Belangrijk*** Als u een niet-verbonden veldobject opmaakt, heeft deze opmaak invloed op waarden die worden ingesteld door ObjectPAL, maar niet op waarden die interactief worden ingevoerd. Als u zeker wilt weten dat alle waarden correct worden opgemaakt, koppelt u bijvoorbeeld de volgende code aan de ingebouwde **changeValue**-methode van het veldobject:

```
method changeValue(var eventInfo ValueEvent)
   try
      eventInfo.setNewValue(Date(eventInfo.newValue()))
   onFail
      message("Ongeldige datum")
      eventInfo.setErrorCode(CanNotDepart)
   endTry
endmethod
```

In dit voorbeeld wordt verondersteld dat u werkt met een datumopmaak. De nieuwe waarde wordt omgezet naar een Date-waarde, die vervolgens wordt toegewezen aan het veldobject. U kunt deze methode ook met andere gegevenstypes gebruiken. Voor een numerieke opmaak zet u de nieuwe waarde bijvoorbeeld om naar een Number.

### Breedte

Specificaties voor de breedte ('W' van Width) bepalen het totale aantal tekens dat een weergegeven of afgedrukte waarde kan bevatten. De breedte-waarden kunnen tussen 1 en 255 liggen. Als u geen breedte opgeeft, wordt de hele waarde uitgevoerd.

Als de opgemaakte uitdrukking een getal is, kunt u ook bepalen hoeveel cijfers rechts van het decimaalteken verschijnen. De totale breedte moet ruimte bevatten voor de gehele waarde, inclusief:

- Alle cijfers links en rechts van het decimaalteken
- Het decimaalteken zelf
- Het getalteken (ongeacht of het wordt weergegeven)
- Alle andere tekens, zoals duizendscheiders en dollartekens

Het aantal cijfers achter het decimaalteken kan maximaal 15 zijn.

Als een getal- of datumwaarde langer is dan de opmaakbreedte, wordt een reeks met asterisken weergegeven. Als de opgegeven breedte niet voldoende is voor het aantal decimale posities in de uitdrukking, wordt de waarde afgerond. Als een alfanumerieke waarde, een logische waarde, een memowaarde, een getal met zwevend decimaalteken of een reeks langer is dan de breedte van de opmaak, wordt de waarde ingekort.

Hier volgen enkele voorbeelden van breedtespecificaties:

```
message(FORMAT("w6","Dit is een test"))   ; geeft Dit is weer
message(FORMAT("w6",1234567)              ; geeft ****** weer)
message(FORMAT("w1",(5 = 5)))             ; geeft True terug, geeft T weer
message(FORMAT("w9.2",1234.567))          ; geeft   1234,57 weer
```

### Uitlijnen

Uitlijning bepaalt de positie van een uitgevoerde waarde binnen de breedte van de opmaak. Als u de uitlijning opgeeft, moet u dus ook een breedte opgeven. Als u dit niet doet, of als de breedte gelijk is aan de lengte van de waarde, heeft de uitlijning geen effect.

Als u geen uitlijning opgeeft, worden reeksen, memowaarden, puntwaarden en logische waarden links uitgelijnd en getallen, tijd- en datumwaarden rechts.

Hier volgen enkele voorbeelden van uitlijnspecificaties:

```
message(FORMAT("w20,ac","Dit is"))        ; resultaat:          Dit is
message(FORMAT("w20,ac","De titel"))      ; resultaat:         De titel
message(FORMAT("w20,ac","van het boek"))  ; resultaat:       van het boek
message(FORMAT("w20,al",123456))          ; resultaat 123456
message(FORMAT("w20,ar",123456))          ; resultaat                123456
```

### Hoofdletters en kleine letters

Met specificaties voor hoofdletters en kleine letters bepaalt u hoe hoofdletters in waarden worden gebruikt. De optie 'CC' maakt bijvoorbeeld van de eerste letter van ieder woord een hoofdletter. Een woord wordt hierbij gedefinieerd als een tekenreeks die loopt tot aan het volgende niet-letter teken.

Hier volgen enkele voorbeelden van specificaties van hoofdletters en kleine letters:

```
message(FORMAT("cu","friet met mayonaise"))     ; resultaat: FRIET MET MAYONAISE
message(FORMAT("cl","DRIE BROODJES GEZOND"))    ; resultaat: drie broodjes
                                                ; gezond
message(FORMAT("cc","sOEP"))                    ; resultaat SOEP
message(FORMAT("cc","'t is weer een mooie " + "zomer"))  ; resultaat: 'T Is
                                                         ; Weer Een Mooie Zomer
```

### Bewerken

Bewerkspecificaties bepalen hoe getallen worden weergegeven. U kunt verschillende bewerkspecificaties combineren achter het voorvoegsel E. Als u een dollarteken wilt gebruiken en duizendtallen wilt scheiden, gebruikt u de opmaakspecificatie "E$C". Hoewel in dit geval het gegevenstype voor de invoer Currency (**$**), Number (**N**), LongInt (**L**) of SmallInt (**S**) kan zijn, kunt u de uitvoer vanwege het dollarteken ($) en de duizendscheider alleen in een alfanumeriek veld plaatsen.

Hier volgen enkele voorbeelden van bewerkspecificaties:

```
x = 34567.89
message(FORMAT("w10.2, e$c", x))          ; resultaat: F34,567.89
message(FORMAT("w10.2, e$ci", x))         ; resultaat: F34.567,89
message(FORMAT("w13.2, e$c", x))          ; resultaat:   F34,567.89
message(FORMAT("w14.2, e$cb, al", x))     ; resultaat: F  34,567.89
message(FORMAT("w15.2, e$cz, al", x))     ; resultaat: F000034,567.89
message(FORMAT("w15.2, e$c*, al", x))     ; resultaat: F****34,567.89
```

De laatste optie wordt vaak gebruikt voor het uitschrijven van cheques, om te voorkomen dat er zonder toestemming cijfers aan een getal worden toegevoegd. Als u een bewerkspecificatie opneemt, gebruik dan in dezelfde aanroep van FORMAT() een specificatie voor de breedte.

Misschien wilt u zowel een tekenspecificatie (beschreven in de volgende paragraaf) als een bewerkspecificatie gebruiken om getallen op te maken.

### Getalteken

Tekenspecificaties bepalen hoe positieve en negatieve waarden worden onderscheiden. Tekenspecificaties hebben alleen betrekking op numerieke waarden en u kunt er maar één tegelijk gebruiken.

Hier volgen enkele voorbeelden van tekenspecificaties:

```
x = -3456.12
message(FORMAT("w8.2, s+", x))          ; resultaat: -3456.12
message(FORMAT("w11.2, e$c, sc", x))    ; resultaat: F3,456.12CR
message(FORMAT("w14.2, e$c*, sp", x))   ; resultaat: (F***3,456.12)
message(FORMAT("w13.2, e$c*, s+", x))   ; resultaat: F-***3,456.12
message(FORMAT("w14.2, e$c*, sd", x))   ; resultaat: F***3,456.12CR
```

DB (debet) en CR (credit) worden voornamelijk gebruikt in boekhoudapplicaties.

U kunt teken- en bewerkspecificaties samen gebruiken. In de drie vorige voorbeelden ziet u de volgorde waarin teken- en bewerkingselementen worden uitgevoerd:

1. Haakje openen, indien opgegeven
2. valutateken, indien opgegeven
3. + of –teken, indien opgegeven
4. Vultekens (spaties, nullen of *), indien nodig en opgegeven
5. Het getal zelf
6. DB of CR, indien opgegeven
7. Haakje sluiten, indien opgegeven

### Datum

Datumspecificaties zorgen voor uitvoer van datumwaarden in een van de datumopmaken van Paradox. Zie de online ObjectPAL Help voor meer informatie. Als u zowel een breedte als datumopmaak opgeeft, moet de breedte groot genoeg zijn voor de datumopmaak. Als dit niet het geval is, verschijnt een reeks asterisken.

Hier volgen enkele voorbeelden van datumspecificaties:

```
da = Date("1/6/92")
message(format("d2", da))       ; resultaat: January 6, 1992
message(format("d7", da))       ; resultaat: 06-Jan-1992
message(format("d11", da))      ; resultaat: 92-01-06
message(format("d7,w5", da))    ; resultaat: *****

message(format("DOLW1",da))     ; resultaat: maa, 01 06, 92

message(format("DOLW2",da))     ; resultaat: maandag, 01 06, 92
message(format("DOLWL",da))     ; resultaat: maandag, 01 06, 92
message(format("DOLWS",da))     ; resultaat: maa 01/06/92
```

```
message(format("DM1",da))      ; resultaat: 1/06/92
message(format("DM2",da))      ; resultaat: 01/06/92
message(format("DM3",da))      ; resultaat: jan/06/92
message(format("DM4",da))      ; resultaat: januari/06/92
message(format("DML",da))      ; resultaat: januari/06/92
message(format("DMS",da))      ; resultaat: 1/06/92

message(format("DO(%M/%D/%Y)",da)) ; resultaat: 01/06/92
message(format("DO(%D/%M/%Y)",da)) ; resultaat: 06/01/92
message(format("DO(%Y/%D/%M)",da)) ; resultaat: 92/06/01
message(format("DO(%D/%Y/%M)",da)) ; resultaat: 06/92/01
message(format("DO(%D%%Y%%M)",da)) ; resultaat: 06%92%01

message(format("DO(%M-%D/%Y)",da)) ; resultaat: 01-06/92
message(format("DO(%D/%M%%Y)",da)) ; resultaat: 06/01%92
message(format("DO(%Y$%D*%M)",da)) ; resultaat: 92$06*01
message(format("DO(%D@%Y!%M)",da)) ; resultaat: 06@92!01
```

### Logisch

Logische specificaties vervangen de standaardwaarden True/False door de logische waarden Yes/No of On/Off. Logische specificaties hebben alleen betrekking op logische waarden.

Bijvoorbeeld:

```
message(FORMAT("LY", (5= 5)))              ; resultaat: Yes
message(format("LT(Goed)",1= 1))           ; resultaat: Goed
message(format("LT(Goed)",1= 2))           ; resultaat: False
message(format("LF(Slecht)",34<12))        ; resultaat: Slecht
```

# Time: klokgegevens

Time-variabelen slaan de tijd op in de vorm uur-minuut-seconde-milliseconde. U kunt de volgende tekens gebruiken als scheidingstekens: tab, spatie, komma (,), koppelteken (-), schuine streep (/), punt (.), dubbele punt (:), en puntkomma (;). U kunt ook geen scheidingsteken gebruiken.

Tijdwaarden moeten worden omgezet (expliciet gedeclareerd). De volgende instructies wijzen bijvoorbeeld aan de Time-variabele *ti* een tijd van tien minuten en veertig seconden over elf 's morgens toe:

```
var ti Time endVar
ti = Time("11:10:40")
```

De aanhalingstekens rond de waarde zijn verplicht. Of een tijd geldig is, hangt af van de huidige tijdopmaak van Windows. Als de tijdopmaak van Windows bijvoorbeeld is ingesteld op een cyclus van twaalf uur (uu:mm:ss), beschouwen methodes in het Time-type uu:mm:ss als een geldige tijdopmaak. Als u de tijdopmaak van Windows vanuit Paradox wilt instellen, gebruikt u de procedure **FormatSetTimeDefault**, die is gedefinieerd voor het System-type.

In Tabel 15-11 ziet u veel gebruikte methodes die zijn gedefinieerd voor het Time-type.

Tabel 15-11 Methodes van het Time-type

| Naam | Beschrijving |
|---|---|
| hour | Haalt de uren uit een Time-waarde |
| milliSec | Haalt de milliseconden uit een Time-waarde |
| minute | Haalt de minuten uit een Time-waarde |
| second | Haalt de seconden uit een Time-waarde |

HOOFDSTUK 16

# Gegevensmodel-objecten

ObjectPAL is een rijke taal waarmee u een grote verscheidenheid aan applicaties kunt ontwikkelen. ObjectPAL is echter op de eerste plaats een databasetaal. In dit hoofdstuk worden de objecten, de types en de methodes behandeld die met tabellen werken. In dit hoofdstuk worden de gegevensmodel-objecten uit Tabel 16-1 behandeld, die toegang geven tot en informatie verschaffen over gegevens die in tabellen zijn opgeslagen.

Tabel 16-1 Gegevensmodel-objecten

| Type | Beschrijving |
|---|---|
| Database | Een verzameling tabellen |
| Query | QBE-queries (query by example) |
| Table | Een tabel |
| TCursor | Een verwijzing naar een tabel die is opgeslagen en wordt gemanipuleerd in het geheugen en niet wordt weergegeven. U gebruikt TCursors om met gegevens in tabellen te werken zonder de gegevens weer te geven. U gebruikt tabelframes of multi-record objecten (beide zijn UIObjecten) om gegevens interactief weer te geven en te manipuleren. |

In dit hoofdstuk wordt ook het volgende beschreven:

- Hoe u een alias gebruikt bij tabellen
- Hoe u een database opent
- Hoe u queries maakt en uitvoert
- Hoe u bewerkingen uitvoert op tabelniveau en tabelattributen opgeeft
- Hoe u Tcursors gebruikt om gegevens te manipuleren op tabel-, record- of veldniveau zonder de tabel weer te geven
- Hoe u velden opgeeft
- Hoe u records zoekt, leest en bewerkt in een tabel

- Hoe u tabellen en TCursors gebruikt om lijsten te programmeren

# Database: een verzameling tabellen

Een Database-variabele verschaft een *handle*, de naam van een variabele die u kunt gebruiken om een database op te geven en met de database te werken. Sommige methodes hebben een Database-variabele als argument en andere methodes werken op de database die door een Database-variabele wordt opgegeven. Als u werkt met Paradox- en dBASE-tabellen, zijn *database* en *directory* synoniemen.

Een *alias* is eigenlijk een pad naar tabellen. Als u werkt met Paradox- en dBASE-tabellen, is een alias een tekenreeks die een directorypad vertegenwoordigt. Met andere woorden, als u een alias gebruikt, geeft u een directory op. Paradox gebruikt de aliassen WORK en PRIV om te verwijzen naar respectievelijk uw werkdirectory en uw privé-directory.

Hoewel databasevariabelen veel lijken op aliassen, worden deze variabelen gebruikt om te verwijzen naar een verzameling tabellen. Aliassen zijn korte aanduidingen voor directories die aangeven waar een tabel is opgeslagen.

## Werken met een alias

Stel dat u wilt werken met bepaalde Paradox-tabellen in de database C:\APPLS\PARADOX\TABELLEN\MASTDATA. In plaats van het hele pad te typen om de tabel te openen, kunt u op een interactieve manier een alias maken (kies 'Bestand | Aliassen') of de **addAlias**-methode van het Session-type gebruiken. De volgende instructie maakt bijvoorbeeld een alias met de naam *mast* voor de directory C:\APPLS\PARADOX\TABELLEN\MASTDATA. (In deze instructie zijn dubbele backslash-tekens nodig, omdat het pad een reeks is die tussen aanhalingstekens staat.)

```
addAlias("mast", "Standard", "C:\\APPLS\\PARADOX\\TABELLEN\\MASTDATA")
```

Als u een alias hebt gemaakt, kunt u deze gebruiken als voorvoegsel om toegang te krijgen tot een bestand. De volgende instructie opent bijvoorbeeld de tabel *Order* in de directory C:\APPLS\PARADOX\TABELLEN\MASTDATA, omdat u de alias *mast* zo hebt gedefinieerd dat deze dit pad vertegenwoordigt.

```
orderTC.open(":mast:Order.db")
```

In de **open**-instructie plaatst u dubbele punten voor en achter de aliasnaam.

Een ander voordeel van het gebruik van aliassen is overdraagbaarheid: in plaats van het directorypad voor elk gebruikerssysteem in harde code op te geven, kunt u de alias eenvoudig opnieuw

definiëren. Stel dat een gebruiker gegevens opslaat in C:\APPLS\PARADOX\TABELLEN\MASTDATA en een andere gebruiker in D:\TABELLEN\MAST. Als u aliassen gebruikt, geeft u alleen de juiste paden op. De rest van uw code hoeft u niet te wijzigen.

***Opmerking*** Als u geen alias opgeeft, gebruikt Paradox WORK.

### *Directorypaden en het gegevensmodel*

Als u een tabel interactief toevoegt aan het gegevensmodel van een formulier, slaat het formulier de tabelnaam als volgt op:

- Tabellen met absolute paden of aliassen (zoals "C:\TEMP\DATA" of ":MYNALIAS:DATA") slaan het pad of de alias precies zo op als deze zijn gedefinieerd.
- Tabellen met ":WORK:" als de alias verwijderen deze alias en slaan alleen de basisnaam van de tabel op.

Als u een formulier opent, zoekt het formulier de tabellen uit het gegevensmodel als volgt:

- Tabellen met padnamen of aliassen gebruiken de volledige padnaam.
- Tabellen zonder pad of alias (dat wil zeggen met alleen een basisnaam) gebruiken het pad of de alias waarin het formulier zich bevindt.

Als u bijvoorbeeld een formulier opent in C:\TEMP (terwijl uw werkdirectory I:\START is) en het gegevensmodel van dat formulier BKBESTEL.DB bevat, is het pad voor de tabel C:\TEMP\BKBESTEL.DB.

Als de tabel een relatief pad heeft, zoals MIJNDIR\BKBESTEL.DB), behandelt het formulier het pad als C:\TEMP\MIJNDIR\BKBESTEL.DB. Relatieve paden werken ook met aliassen. :ALIAS:MIJNDIR\BKBESTEL.DB is dus ook geldig.

***Opmerking*** Het formulier zoekt niet verder naar de tabellen. Als de tabellen zich niet op de plaats bevinden waar het formulier deze verwacht te vinden, verschijnt er een foutmelding.

Als het formulier een tabel vindt, wordt het hetzelfde algoritme gebruikt om opzoektabellen te vinden. Als een opzoektabel is gedefinieerd met een volledig directorypad, wordt dit pad gebruikt om de opzoektabel te vinden. Als de opzoektabel echter geen pad heeft, zoekt Paradox in de directory waarin de hoofdtabel zich bevindt. Als in het vorige voorbeeld "MIJNOPZK.DB" de opzoektabel is, verwacht het formulier deze tabel te vinden in C:\TEMP\MIJNDIR\MIJNOPZK.DB.

Als u een formulier opslaat in een andere directory dan de werkdirectory, gebruikt het systeem dezelfde strategie om volledige namen toe te wijzen aan de tabellen.

## Databases openen

Als u een Paradox-applicatie start, opent Paradox de *standaarddatabase* (de werkdirectory). De standaarddatabase slaat het pad naar de huidige werkdirectory op. Als u alleen met deze tabellen wilt werken, hoeft u geen andere database te openen. Als u wilt werken met tabellen die elders zijn opgeslagen, declareert u een Database-variabele en opent u deze om een handle te maken voor een andere database. (U kunt het volledige pad opgeven van elke tabel die u wilt gebruiken, maar code die Database-variabelen bevat, is gemakkelijker te onderhouden.)

Als u **open** en een alias gebruikt, kunt u opgeven welke database u wilt openen, zoals in het volgende voorbeeld:

```
var
   klantInfo Database
endVar
addAlias("KlantenInfo", "Standard", "D:\\pdoxwin\\tabellen\\klntdata")
klantInfo.open("KlantenInfo") ; opent de KlantenInfo-database
                              ; KlantenInfo moet een geldige alias zijn
```

Paradox heeft nu informatie over twee databases: de standaarddatabase en KlantenInfo. De variabele *klantInfo* is een *handle* voor de database KlantenInfo. Dat wil zeggen dat u *klantInfo* in instructies kunt gebruiken om naar de database KlantenInfo te verwijzen. Stel dat u twee bestanden hebt met de naam ORDER.DB (één in uw werkdirectory en één in KlantenInfo) en dat u wilt weten of deze bestanden tabellen zijn. In het volgende voorbeeld wordt eerst gekeken naar ORDER.DB in de werkdirectory. Vervolgens wordt *klantInfo* als handle gebruikt voor de database KlantenInfo en wordt naar het bestand ORDER.DB in deze database gekeken.

```
var
   klantInfo Database
endVar
addAlias("KlantenInfo", "Standard", "D:\\pdoxwin\\tabellen\\klntdata")
klantInfo.open("KlantenInfo")

if isTable("order.db") then            ; controleer ORDER.DB in de
                                       ; standaarddatabase
   msgInfo("Werk-directory", "ORDER.DB is een tabel.")
endIf

if klantInfo.isTable("order.db") then  ; gebruik klantInfo als een handle voor
                                       ; de database KlantenInfo
msgInfo("KlantenInfo", "ORDER.DB is een tabel.")
endIf
```

Als u **open** gebruikt, maar geen database opgeeft, gaat Paradox ervan uit dat u in de standaarddatabase wilt werken. De volgende code geeft u bijvoorbeeld een handle voor de standaarddatabase, die u

kunt doorgeven aan een eigen methode die een database-handle nodig heeft:

```
var standaardDb Database endVar
standaardDb.open() ; opent de standaarddatabase
```

Als u een handle gebruikt voor de standaarddatabase, wordt de code vaak leesbaarder, vooral als u met meerdere databases tegelijk werkt.

# Query: gegevens opvragen

Een Query-variabele is een handle voor een QBE-query. U kunt ObjectPAL gebruiken om queries te maken en uit te voeren alsof u Paradox interactief gebruikt. U kunt een query uitvoeren via een query-bestand, een query-opdracht of een query-reeks tussen aanhalingstekens. Paradox slaat query-resultaten standaard op in ANTWRD.DB, in de privé-directory van de gebruiker (aangegeven met de alias :PRIV:). U kunt echter ook een andere tabel en een andere directory opgeven.

QBE is een krachtig instrument dat u veel programmeerwerk kan besparen. Stel dat u een database voor werknemers hebt, een functie-omschrijving wilt veranderen van "Manager" in "Chef" en het salaris wilt wijzigen van F 100.000 in F 120.000. U kunt hiervoor ObjectPAL-code schrijven met **scan**-lussen of **if...then...else**-blokken. U verricht dan echter meer werk dan nodig is. Met een eenvoudige WIJZIGIN-query kunt u dezelfde taak uitvoeren.

***Belangrijk*** Voordat u queries maakt en uitvoert met ObjectPAL, moet u vertrouwd zijn met de interactieve Query-Editor. Zie het *Handboek* voor meer informatie over het interactieve gebruik van queries.

Als u een QBE-query (query by example) maakt door Paradox interactief te gebruiken, maakt u een query-formulier dat een voorbeeld geeft van de gewenste gegevens en slaat u het formulier op. Als u een query maakt met ObjectPAL, kunt u het volgende doen:

- Een opgeslagen query uitvoeren
- Een query-opdracht definiëren en deze uitvoeren
- Een query-reeks definiëren en deze uitvoeren met de waarden van andere reeksen en variabelen

Sommige methodes voor het werken met queries zijn gedefinieerd voor het Database-type, omdat u in sommige applicaties de database moet opgeven die de tabellen bevat waarop u een query wilt uitvoeren.

## Query-bestanden

De uitvoering van een query-bestand is zeer eenvoudig te programmeren. Als het query-bestand al is gemaakt en opgeslagen (bijvoorbeeld met behulp van de interactieve Query-Editor), gebruikt u **executeQBEFile** om de query te starten en de resultaten naar een tabel te schrijven. De code in het volgende voorbeeld start de query die is opgeslagen in het bestand NWEORDER.QBE en schrijft de resultaten naar een tabel met de naam NWEORDER.DB in de werkdirectory. (Als u geen tabel opgeeft, worden de resultaten naar ANTWRD.DB in de privé-directory geschreven.)

```
var
   tv TableView
endVar

; de resultaten van de query worden naar nweorder.db geschreven
if executeQBEFile("nweorder.qbe", "nweorder.db") then
   tv.open("nweorder.db")
else
   msgStop("Stop", "Kan de query niet starten.")
endIf
```

U kunt ook een alias gebruiken om een pad op te geven naar de tabel *Antwrd* (of een andere tabel voor de resultaten). De volgende instructie schrijft de resultaten bijvoorbeeld naar NWEORDER.DB in de directory die wordt vertegenwoordigd door de alias *mast* (*mast* moet dan wel ergens zijn gedefinieerd).

```
executeQBEFile("nweorder.qbe", ":mast:nweorder.db")
```

## Query-opdrachten

Een query-opdracht begint met het sleutelwoord **Query** en eindigt met het sleutelwoord **endQuery**. Tussen deze sleutelwoorden geeft u de tabel of de tabellen op waarop u een query wilt uitvoeren, en de velden en de selectiecriteria. Als u een query-opdracht wilt starten, gebruikt u **executeQBE**. Deze methode slaat de resultaten standaard op in ANTWRD.DB in de privé-directory. U kunt echter ook een andere tabel opgeven.

***Opmerking*** U hoeft geen overzicht te geven van alle velden in een tabel. U hoeft alleen een overzicht te geven van de velden die betrekking hebben op de query.

```
mijnQBE = Query

order.db | Naam   | Aantal  |
         | Check  | >10     |

endQuery

executeQBE(mijnQBE) ; slaat resultaten standaard op in :PRIV:ANTWRD.DB
```

De bovenstaande query zoekt in de tabel *Order* de namen van alle klanten die meer dan tien artikelen hebben besteld. (De tabel *Order* heeft meer dan twee velden, maar alleen de twee velden waarin u bent geïnteresseerd, zijn in dit voorbeeld opgegeven).

***Belangrijk*** De lege regel na het sleutelwoord **Query** en de lege regel vóór het sleutelwoord **endQuery** zijn verplicht. Als u een query uitvoert op meer dan één tabel, moet u een lege regel plaatsen tussen de query-opdracht voor elke tabel. U hoeft de kolommen in een query niet uit te lijnen . Als u dat wel doet, is de code echter gemakkelijker te lezen.

Een QBE-query-opdracht kan variabelen, voorbeeldelementen en operatoren bevatten (zie de volgende paragrafen).

## Query-variabelen

U kunt variabelen gebruiken in query-opdrachten door deze te laten voorafgaan door een tilde (~). Als een query-opdracht een tilde-variabele bevat, wordt deze vervangen door de huidige waarde. In het volgende voorbeeld is de naam van de variabele *mijnArtikel*. Het voorbeeld gaat uit van een formulier dat het veldobject *gebrArtikel* bevat. De query zoekt de namen van klanten die het artikel hebben besteld dat in *mijnArtikel* is opgegeven, en slaat de resultaten op in het bestand ARTIKEL.DB in de werkdirectory. Als de query om de een of andere reden mislukt, zorgt de aanroep van de System-procedure **errorShow** ervoor dat er een modaal dialoogvenster verschijnt met informatie over de fout.

```
var
   qopdr2 Query
endVar

mijnArtikel = gebrArtikel.waarde ; haal waarde uit veld 'gebrArtikel' op
                                 ; formulier
qopdr2 = Query

order.db   | Naam   | Artikel            |
           | Check  | Check ~mijnArtikel |    ; tilde-variabele is mijnArtikel

endQuery

if not executeQBE(qopdr2, "artikel.db") then  ; slaat resultaten op in
                                              ; :WORK:ARTIKEL.DB
   errorShow()
endIf
```

Het laatste codeblok is een **if...then**-blok dat controleert of de query met succes is uitgevoerd. Als er een fout optreedt, zorgt de aanroep van de System-procedure **errorShow** ervoor dat er een dialoogvenster verschijnt met de foutcode en de foutmelding. Raadpleeg Hoofdstuk 19 voor meer informatie over fouten en de behandeling van fouten.

U kunt ook uitdrukkingen gebruiken in queries. U plaatst dan een tilde vóór de uitdrukking (zie het volgende voorbeeld). De code in het volgende voorbeeld zoekt de namen van alle werknemers waarvan het salaris hoger is dan F 36.000 (een basiswaarde van F 35.000 waarbij F 1.000 wordt opgeteld in de tilde-uitdrukking).

```
method pushButton(var eventInfo Event)
   var
      qOpdr Query
      basisWaarde LongInt
```

```
   endvar

   basisWaarde = 35000

   qOpdr = Query

        werknmr.db  | Naam  |  Salaris                        |
                    | Check |  Check >~(basisWaarde + 1000) |

           EndQuery

   if not executeQBE(qOpdr) then ; slaat resultaten op in :PRIV:antwrd.db
      errorShow()
   endIf
endmethod
```

Het volgende voorbeeld gebruikt een tilde-variabele die de naam vertegenwoordigt van de tabel waarop een query moet worden uitgevoerd. Het voorbeeld gebruikt ook de **view**-methode die is gedefinieerd voor het String-type, om invoer te krijgen van de gebruiker.

```
var
   qOpdr Query
   tblNaam String
endVar

tblNaam = "Voer tabelnaam in."
tblNaam.view("Welke tabel?") ; Gebruiker typt tabelnaam in dialoogvenster
                             ; en de variabele tblNaam slaat deze op.

qOpdr = Query

        ~tblNaam | Naam  | Telefoon |
                 | Check | Check    |

        endQuery

if not executeQBE(qOpdr) then ; slaat resultaten op in :PRIV:antwrd.db
   errorShow()
endIf
```

## Aliassen

U kunt ook een alias gebruiken om een pad op te geven voor een tabel waarop u een query wilt uitvoeren. De volgende query-reeks is bijvoorbeeld geldig, omdat deze de alias *mast* gebruikt om het pad voor het bestand KLANT.DB aan te geven:

```
var
   qOpdr Query
endVar

qOpdr = Query

          :mast:klant.db | Klantnr.  | Naam  |
                         | _klant    | Check |

          endQuery

if not executeQBE(qOpdr) then
                       ; plaatst resultaten standaard in :PRIV:ANTWRD.DB
```

```
   errorShow()
endIf
```

### Voorbeeldelementen

Als u een voorbeeldelement wilt opgeven, plaatst u een onderstrepingsteken (_) vóór de tekst. U kunt bijvoorbeeld **_klant** typen om het voorbeeldelement *klant* op te geven.

Het volgende voorbeeld gebruikt voorbeeldelementen en het sleutelwoord 'Check' (zie de volgende paragraaf) om de namen van klanten te zoeken in de tabel *Klant* en de artikelen die deze klanten hebben besteld, te zoeken in de tabel *Order*. De lege regels na het sleutelwoord Query, tussen de query-opdrachten van elke tabel en vóór het sleutelwoord endQuery zijn verplicht. Omdat een query-opdracht geen reeks tussen aanhalingstekens is, zijn alleen enkele backslash-tekens toegestaan in de directorypaden.

```
var q Query endVar
q = Query

c:\tabellen\klant.db     | Klantnr. | Naam  |
                         | _klant   | Check |

c:\tabellen\order.db     | Klantnr. | Artikel |
                         | _klant   | Check   |

endQuery

executeQBE(q) ; slaat resultaten op in :PRIV:ANTWRD.DB
```

### Operatoren

Tabel 16-2 geeft een overzicht van alle query-operatoren van Paradox, met inbegrip van gereserveerde symbolen en woorden, en van de rekenkundige operatoren, de vergelijkingsoperatoren, de joker-operatoren, de speciale operatoren, de overzichtsoperatoren en de setvergelijkingsoperatoren. Raadpleeg het *Handboek* voor meer informatie over queries en query-operatoren.

De volgende query gebruikt de sleutelwoorden Check en CheckPlus:

```
var qopdr1 Query endVar
qopdr1 = Query

C:\tabellen\klant.db  | Klantnr.            | Naam      |
                      | Check >1200, <1500  | Check     |
                      | Check >2000         | CheckPlus |

EndQuery

executeQBE(qopdr1, "klant.db") ; plaatst resultaten in Klant.db
```

U kunt een joker-operator (.. of @) gebruiken met een query-variabele. De joker-operator moet dan onderdeel zijn van de query-opdracht en niet van de toewijzing van de variabele. Als u bijvoorbeeld alle ordernummers wilt zoeken die eindigen op 301, wijst u de waarde "301" toe aan een variabele met de naam *Ordnum*. Vervolgens geeft u **..~Ordnum** op in het veld 'Ordernr.' van de query-opdracht.

```
var
   q Query
   Ordnum String
endVar

Ordnum = "301"

q = Query

c:\tabellen\order.db | Ordernr.          |
                     | Check ..~Ordnum |

endQuery

executeQBE(q, "ord301.db") ; slaat de resultaten op in :WORK:ORD301.DB
```

Tabel 16-2 Query-operatoren

| Categorie | Operator | Betekenis |
|---|---|---|
| Gereserveerde woorden | check | Toont unieke veldwaarden uit *Antwrd* |
| | checkPlus | Toont veldwaarden uit *Antwrd* met inbegrip van dubbele |
| | checkDescending | Toont veldwaarden in aflopende volgorde |
| | GroupBy | Geeft een groep op voor set-bewerkingen |
| | VOEGIN | Voegt records in met opgegeven waarden |
| | VERWIJDER | Verwijdert records met opgegeven waarden |
| | WIJZIGIN | Verandert opgegeven waarden in velden |
| | SET | Definieert bepaalde records als een set voor vergelijkingen |
| Rekenkundige operatoren | + | Optellen of alfanumerieke reeks aaneenschakelen |
| | - | Aftrekken |
| | * | Vermenigvuldigen |
| | / | Delen |
| | () | Groepeert operatoren in een query-uitdrukking |
| Vergelijkingsoperatoren | = | Gelijk aan (optioneel) |
| | > | Groter dan |
| | < | Kleiner dan |
| | >= | Groter dan of gelijk aan |
| | <= | Kleiner dan of gelijk aan |
| Joker-operatoren | .. | Elke tekenreeks |
| | @ | Elk teken |
| Speciale operatoren | ZOALS | Lijkt op |

| Categorie | Operator | Betekenis |
|---|---|---|
| | NIET | Stemt niet overeen |
| | LEEG | Geen waarde |
| | VANDAAG | Huidige datum |
| | OF | Geeft OF-voorwaarden op in een veld |
| | , | Geeft EN-voorwaarden op in een veld |
| | ALS | Geeft de naam op van een veld in *Antwrd* |
| | ! | Toont alle waarden in een veld, ongeacht of deze overeenstemmen |
| Overzichtsoperatoren | GEMIDDELD | Gemiddelde van waarden in een veld |
| | TELLING | Aantal waarden in een veld |
| | MIN | Laagste waarde in een veld |
| | MAX | Hoogste waarde in een veld |
| | SOM | Totaal van alle waarden in een veld |
| | ALLE | Berekent overzicht op basis van alle waarden in een groep, met inbegrip van dubbele |
| | UNIEK | Berekent overzicht op basis van unieke waarden in een groep |
| Set-vergelijkings-operatoren | ONLY | Toont records die alleen overeenstemmen met de leden van de gedefinieerde set |
| | GEEN | Toont records die niet overeenstemmen met leden van de gedefinieerde set |
| | ELKE | Toont records die overeenstemmen met alle leden van de gedefinieerde set |
| | EXACT | Toont records die overeenstemmen met alle leden van de gedefinieerde set en niet met andere |

## Query-reeksen

Een query-reeks lijkt op een query, maar met de volgende verschillen:

- Het "**Query...endQuery**"-blok staat tussen dubbele aanhalingstekens.
- Een query-reeks kan worden samengesteld uit kleinere reeksen. Dit is handig als u een query wilt definiëren op basis van een context of de interactie met de gebruiker.
- Een query-reeks kan geen tilde-variabelen bevatten, maar u kunt String-variabelen gebruiken om hetzelfde resultaat te krijgen.

U start een query-reeks met **executeQBEString**. Deze methode slaat de resultaten standaard op in ANTWRD.DB, maar u kunt ook een andere tabel opgeven. In tegenstelling tot een query-opdracht, is een

query-reeks een reeks tussen aanhalingstekens. U moet speciale tekens dus laten voorafgaan door een backslash-teken. (Query-reeksen worden echter niet in een Windows-resource-bestand opgeslagen, omdat deze langer kunnen zijn dan 255 tekens.)

De volgende query-reeks wordt samengesteld door reeksen tussen aanhalingstekens en de String-variabele *qr* aaneen te schakelen. Nieuwe-regeltekens ("\n") geven de verplichte lege regels aan. De query zoekt het klantnummer en de naam van iedere klant met een klantnummer dat groter is dan 100.

```
var
   qr, qreeks String
endVar
qr = "C:\\tabellen\\klant.db | Klantnr.      | Naam      |"

qreeks = "Query" +
       "\n\n" +
        qr + "\n" +
       "| Check >100 | CheckPlus  |" +
       "\n\n"  +
       "endQuery"

executeQBEString(qreeks) ; plaatst resultaten standaard in :PRIV:ANTWRD.DB
```

U kunt geen tilde-variabelen gebruiken in een QBE-reeks. U kunt echter hetzelfde resultaat krijgen door een String-variabele te declareren en deze als onderdeel van de query te gebruiken. Het volgende voorbeeld haalt de waarde van het veld 'onderdeelNaam' uit het eerste record van de tabel *Onderdln* en slaat deze op in de String-variabele *qv*. Vervolgens wordt de query-reeks *qr* samengesteld met reeksen tussen aanhalingstekens en de String-variabele *qv*. De query zoekt de naam van iedereen die het door *qv* opgegeven onderdeel heeft besteld.

```
var
   qr, qv String
   tc TCursor
endVar
tc.open("onderdln.db")
qv = tc.onderdeelNaam ; haal waarde uit veld 'onderdeelNaam' van tabel Onderdln

qr = "Query" +
"\n" + "\n" +
"order.db    | Naam  | Artikel             |\n" +
"            | Check | Check " + qv + " |\n" +
"\n" + "\n" +
"endQuery"

executeQBEString(qr) ; plaatst resultaten standaard in :PRIV:ANTWRD.DB
```

### Aliassen voor het pad van de tabel waarop een query wordt uitgevoerd

U kunt een alias gebruiken om een pad op te geven voor de tabel waarop u een query wilt uitvoeren. De volgende query-reeks is bijvoorbeeld geldig, omdat deze de alias *mast* gebruikt om het pad voor KLANT.DB aan te geven:

```
var
   qReeks String
endVar

qReeks = "Query

          :mast:klant.db    | Klantnr. | Naam  |
                            | _klant   | Check |
          endQuery"

if not executeQBEString(qReeks) then
                        ; plaatst resultaten standaard in :PRIV:ANTWRD.DB
errorShow()
endIf
```

U kunt ook **getAliasPath**, gedefinieerd voor het System-type, gebruiken om te bepalen welk pad een alias vertegenwoordigt. **getAliasPath** geeft een reeks terug die u aan een String-variabele kunt toewijzen en in een query-reeks kunt gebruiken. De volgende code bepaalt bijvoorbeeld het pad van de alias *mast* en wijst het toe aan de variabele *padReeks*. Vervolgens wordt *padReeks* gebruikt als onderdeel van de query-reeks. De dubbele backslash-tekens vóór de tabelnaam zijn verplicht.

```
var
   qReeks, padReeks String
endvar

padReeks = getAliasPath("mast")

   qReeks = "Query" +
            "\n\n" +
             padReeks + "\\klant.db      | Telefoon |
                                        | Check    |\n" +
            "\n\n" +
            "endQuery"

if not executeQBEString(qReeks, "klanttfn.db") then
                        ; schrijft resultaten standaard naar :WORK:KLANTTFN.DB
   errorShow()
endIf
```

# Tabel: een beschrijving van tabulaire gegevens

Een Table-variabele vertegenwoordigt en beschrijft tabulaire gegevens. Een Table-variabele opent echter geen tabellen en geeft ook geen gegevens weer. U kunt een Table-variabele gebruiken om een tabel te beschrijven, zelfs voordat u de tabel zelf hebt gemaakt. Het verschil tussen een Table-variabele en een TCursor is dat een TCursor een verwijzing naar de gegevens is. Het verschil tussen een Table-variabele en een tabelvenster, tabelframe of multi-record object is dat dit objecten zijn die de gegevens weergeven.

U kunt twee soorten handelingen uitvoeren met Table-variabelen en Table-type-methodes:

- U kunt handelingen uitvoeren op tabelniveau, zoals toevoegen, kopiëren, maken, sorteren en indexeren, filters instellen, kolomberekeningen uitvoeren, informatie opvragen over een tabelstructuur enzovoort.

***Opmerking*** U kunt Table-variabelen of methodes van het Table-type niet gebruiken om een tabel te bewerken. In plaats daarvan gebruikt u een Tcursor, een tabelframe of een multi-record object. (Zie "TCursor: een verwijzing naar de gegevens in een tabel", verderop in dit hoofdstuk.)

- U kunt tabelattributen opgeven, zoals filters, indexen en toegangsrechten, voordat u de tabel werkelijk opent. Een Table-variabele gedraagt zich als een controlevariabele voor een tabel en maakt een structuur die de aard van tabulaire gegevens beschrijft of opgeeft.

## Handelingen uitvoeren op tabelniveau

Hoewel het geen methodes of procedures zijn, worden de taalelementen van ObjectPAL voor het maken, indexeren en sorteren van tabellen beschreven in het gedeelte "Table" van de online ObjectPAL Help.

Het volgende eenvoudige voorbeeld laat zien hoe u een tabel maakt:

```
var
   mijnOnderdelen Table
endVar
mijnOnderdelen = CREATE
                   "ONDERDLN.DB"
                   WITH   "Onderdeelnr." : "A20", "Onderdeelnaam" : "A20",
                          "Aantal" : "S"
                   KEY "Onderdeelnr."
                 ENDCREATE
```

Deze code maakt een Paradox-tabel met de naam ONDERDLN.DB. De tabel heeft drie velden: 'Onderdeelnr.', 'Onderdeelnaam' en 'Aantal'. Het veld 'Onderdeelnr.' is een sleutelveld.

Als u een handeling op tabelniveau wilt uitvoeren op een tabel die al bestaat, gebruikt u eerst **attach** om de Table-variabele te associëren met deze tabel. (Er is geen **open**-methode voor het Table-type, omdat u geen Table-variabele kunt openen voor bewerking op record- of veldniveau.) Vervolgens kunt u Table-methodes gebruiken om de beschrijving te manipuleren.

Het volgende voorbeeld declareert de Table-variabele *mijnTabel* en gebruikt **attach** om *mijnTabel* met ORDER.DB te associëren. Vervolgens kijkt **fieldType** naar het tweede veld van de tabel (van links naar rechts geteld, te beginnen met 1, niet met 0). Als veld 2 een datumveld is, telt **cCount** de records met invoer in veld 2 en zoekt **fieldName** de naam van veld 2. De informatie verschijnt vervolgens in een dialoogvenster.

```
var
   mijnTabel  Table                          ; declareer de Table-variabele
   aantDatums LongInt
   vNaam    String
endVar
if mijnTabel.attach("order.db") then         ; associeer mijnTabel met ORDER.DB
   if mijnTabel.fieldType(2) = "Date" then   ; als het tweede veld een
                                             ; datumveld is
      aantDatums = mijnTabel.cCount(2)       ; tel dan de records met invoer
                                             ; in veld 2
      vNaam = mijnTabel.fieldName(2)         ; vraag de naam van veld 2 op
      msgInfo("ORDER.DB", vNaam + " heeft " + String(aantDatums) + " datums.")
   endIf
endIf
```

In een methode kunt u een Table-variabele met meer dan één tabel associëren (en met verschillende types tabellen), maar wel met één tabel tegelijk.

Het volgende voorbeeld declareert de Table-variabele *mijnTbl* en gebruikt **attach** om *mijnTbl* te associëren met de Paradox-tabel ORDER.DB. Vervolgens berekent **cAverage** het gemiddelde van de waarden in het veld 'Aantal'. Daarna koppelt **attach** *mijnTbl* aan de dBASE-tabel VERKOOP.DBF (de Table-variabele wordt automatisch ontkoppeld). De volgende aanroep van **attach** associeert de Table-variabele *bakTbl* met VERKOOP.BAK. Ten slotte maakt **copy** een reservekopie van VERKOOP.DBF en geeft **unlock** de vergrendeling op VERKOOP.BAK vrij.

```
var
   mijnTbl, bakTbl Table
   gemAantal Number
endVar

mijnTbl.attach("order.db", "Paradox")     ; associeer mijnTbl met ORDER.DB
gemAantal = mijnTbl.cAverage("Aantal")

mijnTbl.attach("verkoop.dbf", "dBASE")    ; associeer mijnTbl met VERKOOP.DBF
bakTbl.attach("verkoop.bak", "dBASE")     ; associeer bakTbl met VERKOOP.BAK

mijnTbl.copy(bakTbl)                      ; kopieer VERKOOP.DBF naar VERKOOP.BAK
```

## Tabelattributen opgeven

Andere methodes van het Table-type geven tabelattributen op, zoals welke indexen er moeten worden gebruikt en worden onderhouden en of verwijderde records moeten worden weergegeven. (In het gedeelte "Table" van de online ObjectPAL Help vindt u voorbeelden. hiervan)

***Opmerking*** Deze methodes van het Table-type openen de tabel niet en voeren ook geen controles uit. De methodes definiëren alleen een structuur voor de TCursor-methode **open**.

U gebruikt deze methodes nadat u **attach** hebt gebruikt en voordat u de Table-variabele gebruikt om een TCursor te openen. (TCursors worden in de volgende paragraaf behandeld.) Bijvoorbeeld:

```
var
   ordTbl Table
   ordTC  TCursor
endVar
ordTbl.attach("order.dbf")     ; associeer ordTbl met order.dbf
ordTbl.setIndex("ordx.ndx")    ; stel tabelattributen in
ordTbl.usesIndexes("ord1.ndx", "ord2.ndx")
ordTbl.setReadOnly(Yes)
ordTbl.showDeleted(Yes)

ordTC.open(ordTbl)             ; open de TCursor met de opgegeven attributen
```

***Opmerking*** Methodes die kolombewerkingen uitvoeren (bijvoorbeeld **cAverage** en **cCount**) werken op het gegevensbereik dat wordt opgegeven door de tabelvariabele. De volgende instructies associëren bijvoorbeeld de Table-variabele *klantTbl* met KLANT.DB en roepen vervolgens **cCount** aan om de records te tellen die geen niet-lege waarde in het eerste veld hebben. De resultaten verschijnen in een dialoogvenster. De aanroep van **setFilter** verkleint de tabelbeschrijving door records op te geven waarvan de waarden in het eerste veld tussen 2000 en 4000 liggen. Een tweede aanroep van **cCount** geeft het aantal records terug dat aan deze criteria voldoet. Een nieuw dialoogvenster toont de resultaten.

```
var
   klantTbl Tabel
endVar

klantTbl.attach("klant.db")
msgInfo("Vóór filter", klantTbl.cCount(1)) ; geeft 55 weer

klantTbl.setFilter(2000, 4000) ; geef filtercriteria op
msgInfo("Na filter", klantTbl.cCount(1))   ; geeft 20 weer
```

Zoals u in dit voorbeeld ziet, kunt u kolomfuncties zowel met als zonder filtering van de records gebruiken met een Table-variabele. U kunt kolomfuncties ook gebruiken met TCursor-variabelen.

# TCursor: een verwijzing naar de gegevens in een tabel

Een TCursor is een verwijzing naar gegevens in een tabel, waarmee u gegevens kunt manipuleren op tabel-, record- en veldniveau, zonder dat u de tabel hoeft weer te geven. Een TCursor is geen kloon of kopie van een tabel. Als u de records in een TCursor bewerkt, verandert de onderliggende tabel en vergrendelingen op de tabel hebben invloed op de TCursor.

De TurboBalk bevat geen hulpmiddel waarmee u een TCursor kunt maken, zoals bij tabelframes. Een TCursor is puur een programmeerconstructie en wel de belangrijkste constructie van ObjectPAL voor het werken met tabellen.

Een TCursor en een tabel verhouden zich tot elkaar als een invoegpositie en een tekstverwerkingsdocument. In een

tekstverwerkingsprogramma wijst de invoegpositie telkens naar één letter tegelijk. De invoegpositie kan overal in het document worden geplaatst en geeft de positie aan waar de bewerking plaatsvindt. De TCursor die u voor een tabel opent, wijst naar het huidige record. De TCursor kan eveneens naar elk record in de tabel worden verplaatst en geeft aan welk record wordt bewerkt. U kunt een TCursor ook gebruiken om handelingen op tabelniveau uit te voeren. Tabel 16-3 geeft een overzicht van enkele methodes voor tabellen en TCursors.

Tabel 16-3 Methodes voor tabellen en TCursors

| Taak | Methodes |
|---|---|
| Structuurinformatie opvragen | nRecords, nFields, nKeyFields, fieldName, fieldNo, fieldType, enumTableProperties |
| Verplaatsen | home, end, moveToRecord, nextRecord, priorRecord, skip |
| Positie bepalen | atFirst, atLast, bot, eot, recNo |
| Waarden zoeken | locate, locateCase, locateNext, locatePrior, locatePattern, locateNextPattern |
| Kolomberekeningen uitvoeren | cAverage, cMax, cMin, cNpv, cStd, cSum |
| Werken met records | copyRecord, currRec, initRecord, insertRecord |
| Toegang besturen | lock, unLock, lockRecord, unLockRecord, fieldRights |

Als u een TCursor-variabele declareert en deze naar een tabel laat verwijzen, kunt u de TCursor gebruiken om de tabel te bewerken zonder de tabel weer te geven. Het gebruik van een TCursor lijkt op het gebruik van de afstandsbediening van een TV voor het zoeken van een andere zender: u drukt op een knop van de afstandsbediening en de televisie verandert van kanaal. Op dezelfde manier verandert een record in de onderliggende tabel als u het in een TCursor bewerkt.

U kunt een TCursor op drie manieren naar een tabel laten verwijzen:

- Open de tabel rechtstreeks
- Open de tabel met attributen die door een Table-variabele worden opgegeven
- Associeer een TCursor met een tabelvenster of UIObject dat met een tabel is verbonden

***Opmerking*** Als u een TCursor koppelt aan een UIObject dat gekoppelde tabellen bevat, krijgt de TCursor alleen de structuur en de gegevens van de hoofdtabel.

## Tabel rechtstreeks openen

Als u een TCursor rechtstreeks wilt openen zonder attributen in te stellen, gebruikt u de TCursor-methode **open**. De TCursor verwijst rechtstreeks naar de tabel, ongeacht welke objecten (bijvoorbeeld een tabelframe) zich in het formulier bevinden. Vervolgens kunt u methodes van het TCursor-type gebruiken om informatie te krijgen, de invoegpositie te verplaatsen, waarden te zoeken, waarden op te vragen, waarden in te stellen, berekeningen te maken, de toegang te besturen enzovoort. Dit voorbeeld zorgt ervoor dat *orderTC* naar het eerste record van ORDER.DB verwijst:

```
var
   orderTC TCursor            ; declareer de TCursor
endVar
orderTC.open("order.db")      ; open een invoegpositie in de tabel
```

U kunt nu TCursor-methodes op *orderTC* gebruiken om met de gegevens in ORDER.DB te werken.

Binnen een methode kunt u een TCursor-variabele naar meer dan één tabel laten verwijzen, maar slechts naar één tabel tegelijk. In het volgende voorbeeld verwijst de TCursor-variabele *tcVar* eerst naar DUIKSPUL.DB en vervolgens naar KLANTEN.DB. De TCursor wordt tussendoor automatisch gesloten.

```
var
   tcVar TCursor
   x String
endVar

tcVar.open("duikSpul.db")
x = tcVar.fieldName(1)
x.view()                ; geeft de naam van het eerste veld in DUIKSPUL.DB weer

tcVar.open("klanten.db")
x = tcVar.fieldName(1)
x.view()                ; geeft de naam van het eerste veld in KLANTEN.DB weer
```

## Tabellen openen met attributen

U kunt een Table-variabele gebruiken om tabelattributen als filters, indexen en toegangsrechten, op te geven voordat u een TCursor gebruikt om de tabel te openen. Een Table-variabele gedraagt zich als een controlevariabele en maakt een venster waardoor de TCursor de tabel kan openen, zoals het volgende voorbeeld laat zien:

```
var
   ordTbl Table
   orderTC TCursor
endVar

ordTbl.attach("order.dbf")                ; associeer ordTbl met order.dbf
ordTbl.setIndex("ordx.ndx")               ; stel tabelattributen in
ordTbl.usesIndexes("ord1.ndx", "ord2.ndx")
ordTbl.setReadOnly(Yes)
ordTbl.showDeleted(Yes)

orderTC.open(ordTbl)   ; open de TCursor met de opgegeven attributen
```

***Opmerking*** In het volgende voorbeeld is de eerste instructie niet gelijk aan de tweede:

```
ordTC.open(ordTbl)

ordTC.open("order.dbf")
```

De eerste instructie opent ORDER.DBF met de attributen die in de Table-variabele *ordTbl* zijn opgegeven. De tweede instructie opent ORDER.DBF rechtstreeks, zonder attributen op te geven.

## TCursor associëren met een tabelweergave of een UIObject

U kunt de **attach**-methode van het TCursor-type gebruiken om een TCursor te associëren met de tabel die wordt weergegeven in een tabelvenster. (Het TableView-type wordt behandeld in Hoofdstuk 14). De volgende instructies declareren bijvoorbeeld een TCursor-variabele met de naam *ordTC* en roepen vervolgens **attach** aan om *ordTC* te associëren met de tabel *Order* die wordt weergegeven in een tabelvenster dat is geopend met de TableView-variabele *ordTV*:

```
var
   ordTC TCursor
   ordTV TableView
endVar

if ordTV.open("order.db") then    ; open een tabelvenster
   ordTC.attach(ordTV)            ; associeer een TCursor met de tabel
else
   msgStop("Stop", "Kan de tabel niet openen.")
endIf
```

Als een formulier een UIObject bevat (bijvoorbeeld een veld, een tabelframe of een multi-record object) dat is verbonden met een tabel, kunt u de **attach**-methode van het TCursor-type gebruiken om een TCursor te associëren met het UIObject en daardoor met de onderliggende tabel. Stel dat een formulier een tabelframe bevat met de naam *VERKOOP*. Dit tabelframe is verbonden met VERKOOP.DB. De volgende instructies laten *verkoopTC* verwijzen naar VERKOOP.DB:

```
var verkoopTC TCursor endVar
verkoopTC.attach(VERKOOP)
```

***Opmerking*** In deze paragraaf wordt ervan uitgegaan dat u enigszins vertrouwd bent met tabelvensters, tabelframes en multi-record objecten, zoals beschreven in het *Handboek*. Zie ook Hoofdstuk 13 en Hoofdstuk 14 van deze handleiding.

Een TCursor krijgt de gegevens en de structuur uit de onderliggende tabel en niet uit de weergave. Stel dat een formulier één veld bevat, dat is verbonden met het veld 'Telefoon' in de tabel *Klant*. De volgende instructie associeert de TCursor *klantTC* met alle velden in de tabel *Klant* en niet alleen met het weergegeven veld 'Telefoon'.

```
klantTC.attach(Telefoon)
```

Als u een TCursor koppelt aan een tabelvenster of een UIObject, weet de TCursor niets over gegevens die niet zijn doorgevoerd. Het is dus mogelijk dat een record is veranderd na de aanroep van **attach**. Rotatie van kolommen in een tabelweergave heeft geen invloed op de veldvolgorde zoals de TCursor die kent.

### *Koppelen aan niet-gekoppelde tabellen*

Afbeelding 16-1 laat zien hoe u TCursors kunt koppelen aan UIObjecten als het UIObject is verbonden met één niet-gekoppelde tabel. In de afbeelding zijn drie formulieren afgebeeld. Elk formulier is verbonden met KLANT.DB. In alle gevallen associeert de koppeling van een TCursor met een UIObject de TCursor met alle velden in KLANT.DB.

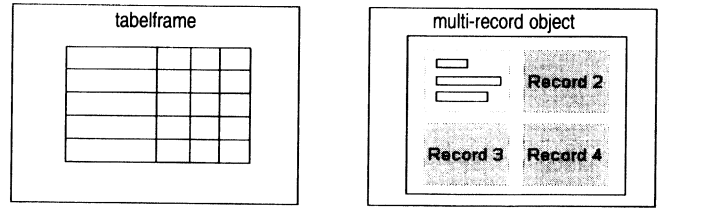
In de eerste twee formulieren, die respectievelijk een tabelframe en een multi-record object bevatten, zorgen de volgende instructies ervoor dat de TCursor *klantTC* naar de tabel *Klant* verwijst. (Als een tabelframe of multi-record object met een tabel is verbonden, neemt het standaard de naam van die tabel over. Het tabelframe en het multi-record object uit de afbeelding zijn verbonden met KLANT.DB en krijgen dus de naam KLANT.)

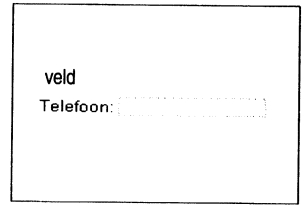
```
var klantTC TCursor endVar
klantTC.attach(KLANT)    ; koppel klantTC aan het UIObject KLANT
```

In het derde formulier, dat een veldobject bevat dat is verbonden met het veld 'Telefoon' van de tabel *Klant*, zorgen de volgende instructies voor hetzelfde resultaat: *klantTC* verwijst naar de tabel *Klant* en niet alleen naar het veld 'Telefoon'. (Als een veldobject is verbonden met een veld in een tabel, krijgt het standaard de naam van dat veld. In de afbeelding is het veldobject verbonden met het veld 'Telefoon' van de tabel *Klant*. Het veldobject krijgt dus de naam 'Telefoon'.)

```
var klantTC TCursor endVar
klantTC.attach(Telefoon)       ; associeer klantTC met het UIObject 'Telefoon'
```

Afbeelding 16-1 Een TCursor koppelen aan een UIObject dat is verbonden met een niet-gekoppelde tabel




Het tabelframe en het multi-record object in de twee bovenstaande formulieren zijn verbonden met KLANT.DB. De volgende instructies associëren de TCursor met de onderliggende tabel:

```
var klantTC TCursor endVar
klantTC.attach(KLANT)
```

Het veldobject in het formulier links is verbonden met het veld 'Telefoon' van KLANT.DB. Deze instructies doen hetzelfde: zorgen dat de TCursor naar KLANT.DB verwijst:

```
var klantTC TCursor endVar
klantTC.attach(Telefoon)
```

## Koppelen aan gekoppelde tabellen

Afbeelding 16-2 laat zien hoe u TCursors koppelt aan UIObjecten als dit UIObject een tabelframe of multi-record object is dat is verbonden met gekoppelde tabellen.

**Opmerking** Als u een TCursor koppelt aan een tabelframe of multi-record object dat is verbonden met gekoppelde tabellen met een één-waarde detailtabel, krijgt de TCursor alleen de structuur en de gegevens van de hoofdtabel.

Afbeelding 16-2 toont twee formulieren. Het gegevensmodel van het eerste formulier, dat een tabelframe bevat, geeft een één→één koppeling op tussen WERKNMR.DB en AFDELING.DB, met WERKNMR.DB als de hoofdtabel. Dit tabelframe toont gekoppelde records uit de tabel *Werknmr* en de tabel *Afdeling*. De volgende instructies associëren de TCursor-variabele *dezeTC* met het UIObject, WERKNMR. *dezeTC* verwijst dus *alleen* naar de tabel *Werknmr*. Dit komt niet doordat het object WERKNMR heet, maar doordat WERKNMR.DB de hoofdtabel is.

```
var dezeTC TCursor endVar
dezeTC.attach(WERKNMR)
```

In het tweede formulier, dat een multi-record object met de naam KLANT en een tabelframe met de naam ORDER bevat, geeft het gegevensmodel een één→meer koppeling op tussen KLANT.DB en ORDER.DB, met KLANT.DB als de hoofdtabel. Net als in het vorige

voorbeeld associëren deze instructies *dezeTC* met het object met de naam KLANT. *dezeTC* verwijst dus alleen naar de tabel *Klant*:

```
var dezeTC TCursor endVar
dezeTC.attach(KLANT) ; associeer dezeTC met MRO KLANT
```

De volgende instructies associëren de TCursor-variabele *dezeTC* met het object ORDER. *dezeTC* verwijst dus naar de tabel *Order*:

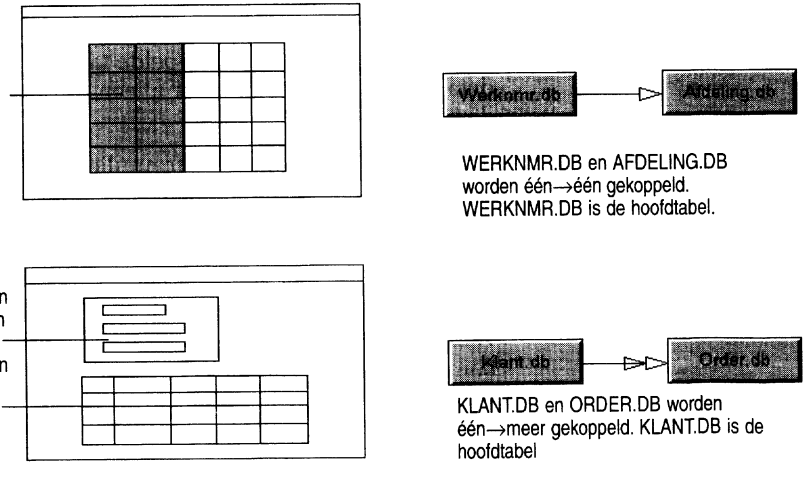
```
var dezeTC TCursor endVar
dezeTC.attach(ORDER) ; associeer dezeTC met het tabelframe ORDER
```

Deze **attach**-instructie associeert *dezeTC* met de tabel *Order* die zich in het tabelframe bevindt en maakt een koppeling met de detail-set van het huidige hoofdrecord. Stel dat het eerste record informatie bevat over een klant die drie bestellingen heeft geplaatst. In dat geval zouden de volgende instructies een **3** weergeven op de statusbalk.

```
var dezeTC TCursor endVar
dezeTC.attach(ORDER)
message(dezeTC.nRecords()) ; geeft 3 weer
```

Als u de invoegpositie verplaatst naar het record van een klant die zeven bestellingen heeft geplaatst, zouden dezelfde instructies een **7** weergeven op de statusbalk, omdat de TCursor is gekoppeld aan een andere detail-set.

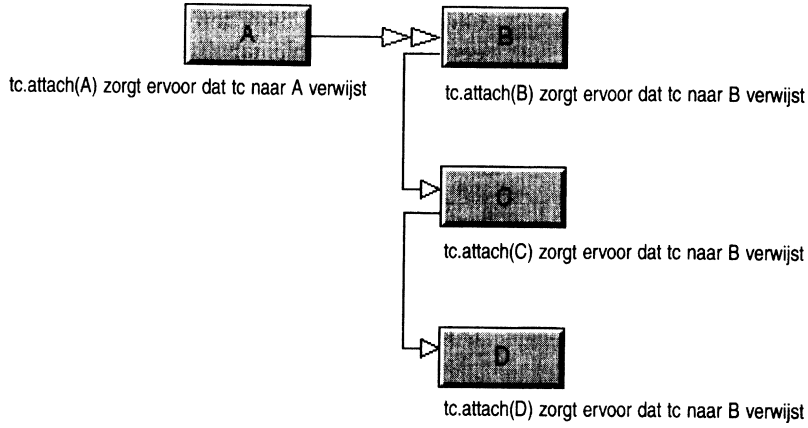
Afbeelding 16-2 Een TCursor koppelen aan een UIObject dat is verbonden met gekoppelde tabellen




*Koppelen aan meerdere gekoppelde tabellen*

Afbeelding 16-3 toont het gegevensmodel van tabellen die zijn gekoppeld in een één→meer→één→één-relatie. De afbeelding geeft ook een overzicht van vier **attach**-instructies en geeft aan naar welke tabel de TCursor verwijst na elke aanroep van **attach**.

Afbeelding 16-3 Een TCursor koppelen met gekoppelde tabellen




# TCursors en tabellen

In deze paragraaf wordt beschreven hoe u TCursors gebruikt bij het werken met tabellen. Verder worden in deze paragraaf voorbeelden getoond van de methodes uit Tabel 16-3 en beschrijvingen gegeven van veel voorkomende taken die u met TCursors kunt uitvoeren.

## Bewerkingen met TCursors

Als u gegevens in een TCursor wilt veranderen, gebruikt u **edit**. Als u alle veranderingen hebt aangebracht, voert u deze door in de onderliggende tabel door het record te verlaten of **postRecord** aan te roepen. Als u uw wijzigingen niet wilt bewaren, gebruikt u **cancel**. Veranderingen worden voor één record tegelijk verwerkt.

Als u klaar bent met de bewerking, gebruikt u **endEdit** om de bewerkmodus te verlaten. Als u klaar bent met de TCursor, gebruikt u **close**.

Het volgende voorbeeld opent een TCursor voor de tabel *Verkoop* en activeert de bewerkmodus voor deze tabel. Vervolgens wordt in het veld 'BedrNaam' van de tabel gezocht naar de waarde "Morland" en wordt deze waarde veranderd in "Borland". Daarna wordt de bewerkmodus voor de tabel beëindigd en wordt de TCursor gesloten, waardoor de associatie van de TCursor met de tabel wordt opgeheven.

```
var tc TCursor endVar
tc.open("verkoop.db")
tc.edit()
if tc.locate("BedrNaam", "Morland") then
   tc.BedrNaam = "Borland"
```

```
endIf
tc.endEdit()
tc.close()
```

## Velden opgeven

ObjectPAL biedt meerdere mogelijkheden om velden op te geven in een TCursor, Table, tabelframe, TableView of multi-record object. Als u de veldnaam kent, kunt u de puntnotatie gebruiken. De volgende code wijst bijvoorbeeld de waarde van het veld 'Aantal' toe aan de variabele *mijnAantal*:

```
var
   tc TCursor
   mijnAantal LongInt
endVar
tc.open("order.db")
mijnAantal = tc.Aantal  ; geef het veld 'Aantal' op
```

U kunt op dezelfde manier een waarde toewijzen aan het veld 'Aantal':

```
tc.Aantal = 120
```

Veldnamen zijn niet aan dezelfde beperkingen gebonden als ObjectPAL-variabelen. Als u werkt met een veld met de naam 'Klant Kredietnummer', dat spaties bevat, plaatst u de naam tussen dubbele aanhalingstekens:

```
mijnAantal = tc."Klant Kredietnummer"
```

U kunt ook variabelen en uitdrukkingen gebruiken om velden op te geven, door deze tussen haakjes te plaatsen. Bovendien kunt u een veld met een nummer opgeven, waarbij u van links naar rechts telt, bijvoorbeeld:

```
var
   tc, andereTC TCursor
   s String
   w AnyType
endVar
tc.open("order.db")

s = "Klant Kredietnummer"
w = tc.(s)                          ; vraagt Klant Kredietnummer op
w = tc.("Klant " + "Kredietnummer") ; doet hetzelfde
w = tc.(2)                          ; vraag de waarde van het tweede veld van
                                    ; links op
w = tc.(mijnMethode())              ; mijnMethode() moet een waarde teruggeven
                                    ; die een veld aangeeft

andereTC.open("andere.db")
w = tc.(andereTC.eenVeld)  ; de waarde van andereTC.eenVeld geeft een veld op
                           ; als andereTC.eenVeld bijvoorbeeld "Aantal" is,
                           ; dan is w tc.Aantal
```

## Waarden uit een tabel halen

De syntaxis voor het lezen van een waarde uit een veld luidt:

*waardeVar = tabelVar.veldID*

*veldID* bevat de veldnaam of een geheel getal dat de positie van het veld in de onderliggende tabel vertegenwoordigt (van links naar rechts geteld), of een uitdrukking die de veldnaam of veldpositie oplevert.

De volgende code geeft een voorbeeld van beide methodes:

```
var
   tc TCursor
   onderdeelNaam String
   aantal LongInt
endVar

if tc.open("onderdln.db") then
   onderdeelNaam = tc."Onderdeelnaam" ; leest gegevens uit het veld
"Onderdeelnaam"
   aantal = tc.(4)                    ; leest uit veld 4 (vierde van links)
endIf
```

## Records toevoegen aan een tabel

Als u records wilt toevoegen aan een tabel, opent u de tabel, activeert u de bewerkmodus, voegt u een record in, plaatst u waarden in het record en beëindigt u de bewerkmodus. De code ziet er als volgt uit:

```
var
   tc TCursor
   endvar
if tc.open("onderdln.db") then      ; open de tabel
   tc.edit()                        ; activeer de bewerkmodus
   tc.insertRecord()                ; voeg een leeg record in

   tc."Onderdeelnaam" = "Tandwiel" ; plaats waarden
   tc."Aantal" = 348
   tc.endEdit()                     ; beëindig bewerkmodus
endIf
```

## Waarden veranderen

Als u waarden in een tabel wilt veranderen, luidt de syntaxis:

*tabelVar.veldID = nieuweWaarde*

*veldID* is de veldnaam of het veldnummer (velden zijn van links naar rechts genummerd vanaf 1). De volgende code doorzoekt het veld 'Onderdeelnaam' in ONDERDLN.DB en vervangt "tandwiel" steeds door "TurboTandwiel".

```
var
   tc TCursor
   nweNaam, oudeNaam, tabelNaam, veldNaam String
endVar

oudeNaam = "tandwiel"
nweNaam = "TurboTandwiel"
tabelNaam = "onderdln.db"
veldNaam = "Onderdeelnaam"
```

```
if tc.open(tabelNaam) then
      tc.edit()
      scan tc for tc.veldNaam = oudeNaam
      tc.veldNaam = nweNaam ; deze regel verandert de veldwaarde
      tc.endEdit()
   else
      msgStop("Stop", "Kan niet vinden " + oudeNaam)
   endIf
else
   msgStop("Stop", "Kan niet openen " + tabelNaam)
endIf
```

# TCursors en UIObjecten

In deze paragraaf worden enkele voorbeelden gegeven van het gebruik van TCursors en UIObjecten. Het eerste voorbeeld vergelijkt de tijd die nodig is om record voor record een tabel te doorlopen met een TCursor en met een tabelframe. Het tweede voorbeeld laat zien hoe u TCursors met UIObjecten gebruikt om de uitvoering te verbeteren als u een tabel doorzoekt.

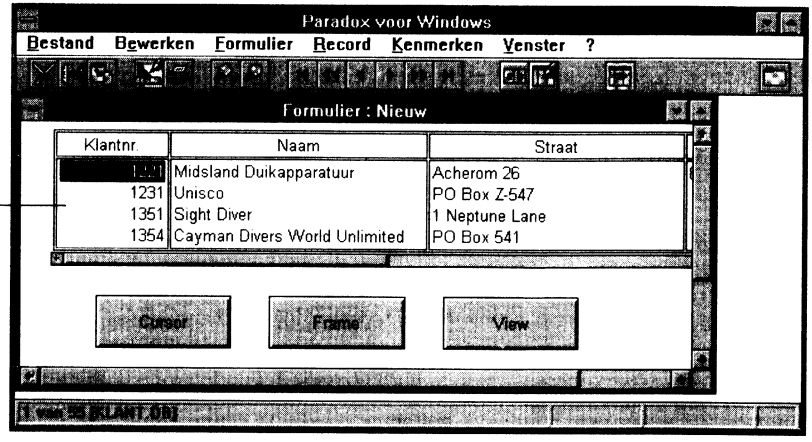
## Records in een tabel doorlopen

Als u TCursors en tabelframes samen gebruikt, kunt u gegevens achter de schermen verwerken, voordat de resultaten aan de gebruiker worden getoond. (Een tabelframe is een UIObject.) Dit werkt vaak beter. De belangrijkste methodes zijn de **attach**-methode van het TCursor-type , die een TCursor met een tabelframe verbindt, en de **moveTo**-methode van het UIObject-type, die het tabelframe afstemt op de TCursor.

Afbeelding 16-4 toont een formulier met een tabelframe (*KLANT*) en drie knoppen. Het tabelframe is verbonden met een tabel, *Klant.db* genaamd. (Als een UIObject is verbonden met een tabel, neemt het object standaard de naam van de tabel over. De naam van het tabelframe is dus *KLANT*, omdat het frame is verbonden met de tabel *Klant*.

De drie knoppen (*gebrCursor*, *gebrFrame* en *gebrWeergave*) hebben dezelfde functie: stap voor stap de records doorlopen en het laatste record in de tabel weergeven. De uitvoering van de methodes voor *gebrFrame* en *gebrWeergave* duurt echter langer, omdat deze, na elke verplaatsing naar een volgend record, het scherm bijwerken. De methode voor *gebrCursor* werkt het scherm in één keer bij en is daardoor veel sneller klaar.

Afbeelding 16-4 Tabelframe, TableView en TCursor




Dit tabelframe is verbonden met Klant.db en heeft daardoor standaard de naam KLANT.

## Methodes koppelen aan knoppen

De volgende methodes worden gekoppeld aan de knoppen *gebrCursor*, *gebrFrame* en *gebrWeergave*.

*gebrCursor* De volgende methode worden gekoppeld aan *gebrCursor*. Deze methode bereikt het laatste record sneller, omdat het scherm maar één keer wordt bijgewerkt.

```
method pushButton (var eventInfo Event)
   var
      i SmallInt
      tc TCursor
      start, stop, delta LongInt
   endVar
   start = CPUClockTime()
   tc.attach(klant)              ; verbindt invoegpositie met tabelframe 'klant'
   for i from 1 to tc.nRecords()
      tc.nextRecord()            ; doorloopt de records
   endFor
   klant.moveToRecord(tc)        ; stemt invoegpositie en frame op elkaar af
                                 ; om het laatste record weer te geven
   stop = CPUClockTime()
   delta = stop - start
   msgInfo("TCursor-tijd", delta)
endMethod
```

*gebrFrame* De volgende methode wordt gekoppeld aan *gebrFrame* en is langzamer dan *gebrCursor*, omdat deze methode het scherm na elke verplaatsing bijwerkt.

```
method pushButton (var eventInfo Event)
   var
      i SmallInt
      start, stop, delta LongInt
   endVar
   start = CPUClockTime()
   for i from 1 to klant.nRecords() ; naam van het frame is standaard 'KLANT'
      klant.nextRecord()
```

```
   endFor
   stop = CPUClockTime()
   delta = stop - start
   msgInfo("Tabelframe-tijd", delta)
endmethod
```

*gebrWeergave*

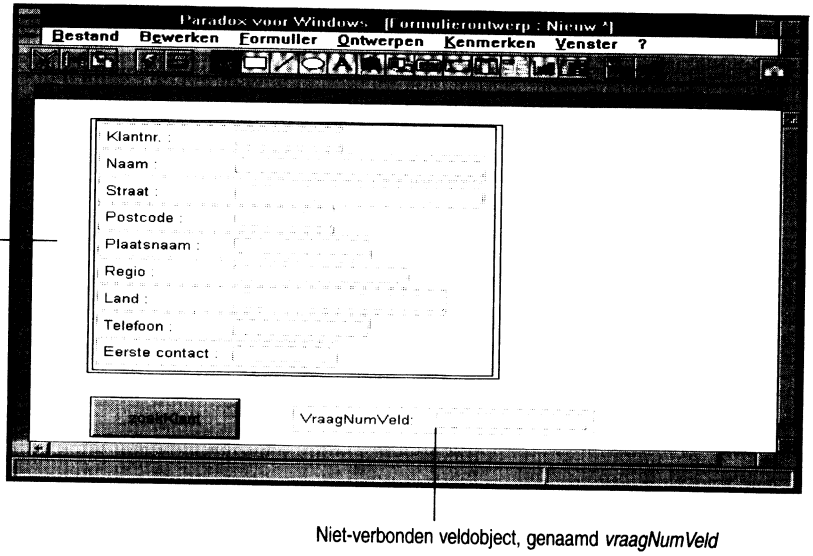
De volgende methode wordt gekoppeld aan *gebrWeergave* en is ook langzamer dan *gebrCursor*, omdat deze methode het scherm bijwerkt na elke verplaatsing.

```
method pushButton (var eventInfo Event)
   var
      klantTV TableView
      i SmallInt
      start, stop, delta LongInt
   endVar
   klantTV.open("klant.db")
   start = CPUClockTime()
   for i from 1 to klantTV.nRecords ; gebruikt TableView-kenmerk, geen methode
      klantTV.rowNo = i             ; gebruikt kenmerk 'rowNo' om naar rij i in
                                    ; tabelvenster te gaan
   endFor
   stop = CPUClockTime()
   delta = stop - start
   msgInfo("Tabelvenstertijd", delta)
endMethod
```

## Tabel doorzoeken

Afbeelding 16-5 toont een formulier met een multi-record object (*KLANT*), dat is verbonden met de tabel *Klant* en is gedefinieerd voor de weergave van één record horizontaal en één record verticaal, een niet-verbonden veldobject (*VraagNumVeld*) en een knop (*zoekKlant*). Typ een getal in het veldobject en klik vervolgens op de knop. Als uw getal overeenstemt met een waarde in het veld 'Klantnr.' van de tabel *Klant*, wordt het overeenkomende record weergegeven in het multi-record object.

Afbeelding 16-5 Een TCursor gebruiken met een multi-record object




De **pushButton**-methode van *zoekKlant* behandelt de verwerking.

```
method pushButton(var eventInfo Event)
var
   klantTC  TCursor
   klantNum String
endVar

klantNum = VraagNumVeld.value
klantTC.attach(klant)              ; associeer klantTC met de tabel Klant

if klantTC.locate("Klantnr.", klantNum) then  ; als de waarde zich in de
TCursor bevindt
   klant.moveToRecord(klantTC)                ; toon het record
   klant.Klantnr_.moveTo()                    ; verplaats de markering naar het
                                              ; veld
else
   msgInfo("Sorry", "Kan niet vinden " + klantNum)
endIf

endmethod
```

Deze methode gebruikt de TCursor om zonder vertraging door de weergave te zoeken en geeft de resultaten vervolgens weer in het multi-record object. U kunt dezelfde methode gebruiken bij tabelframes.

***Opmerking*** U kunt deze taken uitvoeren zonder TCursors. Het UIObject-type bevat methodes (bijvoorbeeld **locate)** die u kunt gebruiken bij tabelframes en multi-record objecten (zie Hoofdstuk 13).

### moveToRecord en moveTo

Twee instructies uit het vorige voorbeeld roepen de methodes **moveToRecord** en **moveTo** van het UIObject-type aan. Deze instructies hebben een verschillende syntaxis en geven een verschillend eindresultaat.

*moveToRecord*

De instructie **moveToRecord** laat het multi-record object *klant* het record weergeven waarnaar door de TCursor *klantTC* wordt verwezen.

```
klant.moveToRecord(klantTC)
```

Deze instructie stemt het multi-record object en de TCursor op elkaar af.

**Opmerking**

De **moveToRecord**-methode is alleen geldig als het UIObject en de TCursor met dezelfde tabel zijn geassocieerd. Als een object met de tabel *Klant* is verbonden en een TCursor met de tabel *Order*, werkt de volgende instructie niet:

```
klant.moveToRecord(orderTC)
```

Deze instructie werkt ook niet als het UIObject niet-verbonden is. Als het object *KLANT* niet is verbonden met de tabel *Klant*, kan het niet naar een TCursor worden "verplaatst".

*moveTo*

De **moveTo**-instructie uit het volgende voorbeeld heeft geen invloed op de TCursor:

```
klant.Klantnr_.moveTo()
```

Deze instructie verplaatst de markering naar het veldobject *Klantnr_* in het tabelframe *KLANT* en maakt het veldobject actief. U kunt de **moveTo**-methode bij elk gebonden of niet-verbonden UIObject gebruiken.

### moveToRecord en TableView

Het TableView-type kent ook een **moveToRecord**-methode die op dezelfde manier werkt als de **moveToRecord**-methode die eerder is besproken. De volgende instructies koppelen bijvoorbeeld een TCursor aan een tabelvenster, zoeken een record in de TCursor en stemmen het tabelvenster af op de TCursor:

```
var
   klantTC TCursor
   klantTV TableView
endVar

klantTV.open("klant.db")                    ; open een tabelvenster
klantTC.attach(klantTV)                     ; associeer TCursor met het
                                            ; tabelvenster
if klantTC.locate("Naam", "Ouwens") then    ; Zoek de TCursor voor een waarde.
   klantTV.moveToRecord(klantTC)            ; Als de waarde wordt gevonden,
else                                        ; stem tabelvenster dan af op TCursor
   msgInfo("Niet gevonden.", "Kan Ouwens niet vinden.")
endIf
```

Raadpleeg de online ObjectPAL Help voor meer informatie over **moveToRecord**.

### *reSync*

Hoewel **moveToRecord** een UIObject afstemt op een TCursor, verandert deze instructie de tabelbeschrijving van het UIObject niet. Het UIObject "verplaatst zich" naar het huidige record van de TCursor, ongeacht eventuele filters, indexen enzovoort, die voor de TCursor zijn gedefinieerd. U Gebruikt de **reSync**-methode van het UIObject-type om een UIObject af te stemmen op een TCursor en rekening te houden met de tabelbeschrijving van de TCursor.

Stel dat een formulier een tabelframe bevat dat is verbonden met de tabel *Klant*. De volgende code associeert een TCursor-variabele met het tabelframe en roept daarna **setFilter** aan om de tabelbeschrijving van de TCursor te veranderen. De aanroep van **reSync** zorgt er vervolgens voor dat het tabelframe alleen de gefilterde gegevens weergeeft. In dit voorbeeld worden records weergegeven van klantnummers tussen 2000 en 4000:

```
var
   klantTC TCursor
endVar
klantTC.attach(KLANT)
klantTC.setFilter(2000, 4000)
KLANT.reSync(klantTC)
```

***Opmerking*** Net als **moveToRecord** werkt **reSync** alleen als het UIObject en de TCursor zijn geassocieerd met dezelfde tabel. Zie ook de uitleg van **attach** bij gekoppelde en niet-gekoppelde tabellen, eerder in dit hoofdstuk.

## Tabellen en TCursors gebruiken om lijsten te programmeren

Deze paragraaf behandelt twee methodes voor het programmeren van afrollijsten (zie Afbeelding 16-6). De eerste methode is snel en gemakkelijk. De tweede methode is iets moeilijker, maar geeft meer controle over de lijst.

***Opmerking*** De technieken die hier worden beschreven, werken zowel voor afrollijsten als voor andere lijsten.

De snelste en gemakkelijkste manier om waarden toe te wijzen aan een afrollijst is via het kenmerk 'DataSource'. Als u bijvoorbeeld een lijst wilt maken van telefoonnummers die in de tabel *Klant* zijn opgeslagen, gaat u als volgt te werk:

1. Maak een leeg, niet-verbonden formulier.

2. Plaats een veldobject en stel het weergavetype in op 'Afrol-bewerken' (of 'Lijst', als u wilt). Als het dialoogvenster 'Lijst definiëren' verschijnt, kiest u 'OK' om het te sluiten.

3. Kies 'Formulier | Objectenschema' terwijl het veldobject is geselecteerd, om het schema van de lijst weer te geven. Een lijst is een uit twee delen samengesteld object: een veldobject en een lijstobject. Koppel code die werkt met de waarden in de lijst met het lijstobject. Koppel vervolgens met het veldobject de code die werkt met de waarden die in het veld worden weergegeven.

4. Inspecteer het lijstobject in het Objectenschema en kies 'Methodes' om het methodevenster te openen.

5. De normale plaats om code voor de lijst te koppelen is de **open**-methode van het lijstobject. U kunt de code echter ook aan andere methodes koppelen en zelfs aan andere objecten als dat nodig is. Koppel de volgende code aan de ingebouwde **open**-methode van het lijstobject:

```
method open (var eventInfo Event)
   self.DataSource = "[klant.telefoon]" ; maak een lijst van het veld
                                        ; 'Telefoon' in tabel Klant
endMethod
```

6. Start het formulier.

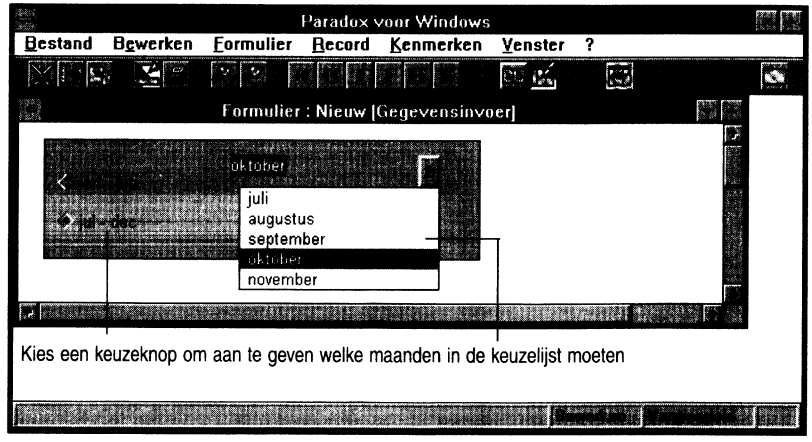
De volgende code maakt een lijst met de waarden in het veld 'Telefoon' van KLANT.DB. (De backslash-tekens vóór de binnenste dubbele aanhalingstekens, zijn verplicht.)

```
self.DataSource = "[\"klant.db\".telefoon]"
; backslash-tekens zijn verplicht voor aanhalingstekens in een reeks
```

Het volgende voorbeeld laat zien hoe u afrollijsten met een invoervak, zoals in Afbeelding 16-6, kunt programmeren. U kunt dezelfde methodes gebruiken als u normale lijsten wilt maken.

Als u dit voorbeeld volgt, maakt u een tabel met de maanden van het jaar en een formulier dat twee velden bevat: het ene veld is gedefinieerd als een paar keuzeknoppen en het andere als een afrol-bewerklijst. De afrol-bewerklijst toont de namen van zes maanden (die uit de tabel zijn gelezen), zoals aangegeven door de keuzeknoppen.

Afbeelding 16-6 Een afrol-bewerklijst




## Tabel maken

1. Maak een Paradox-tabel met een alfanumeriek veld zonder sleutel dat zestien tekens lang is (A16). Noem het veldobject *maandNaam* en noem de tabel *Maanden*.
2. Voer de maanden van het jaar in de juiste volgorde in de tabel *Maanden* in. (De tabel heeft geen sleutel. Hierdoor verschijnen de maanden in kalendervolgorde en niet in alfabetische volgorde.)
3. Sluit de tabel.

Maak vervolgens een leeg, niet-verbonden formulier

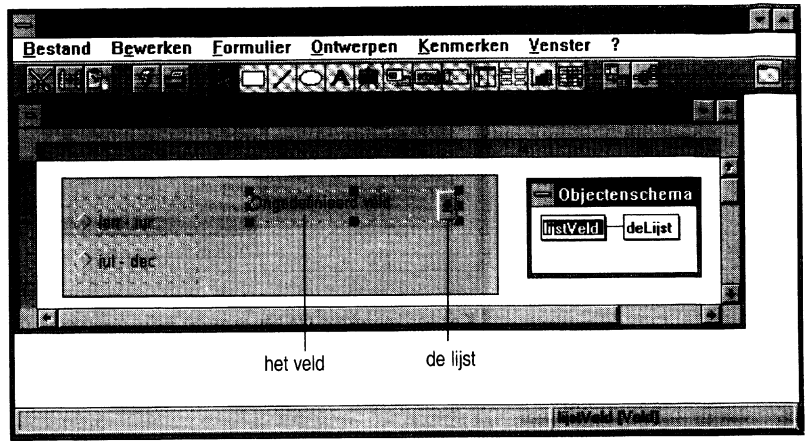
## Formulier maken

*Plaats twee veldobjecten op het formulier.*

1. Kies 'Bestand | Nieuw | Formulier' en kies 'OK' in de dialoogvensters om een leeg, niet-verbonden formulier te maken.
2. Kies het Veld-hulpmiddel op de TurboBalk en plaats een veldobject in het midden van het formulier. Inspecteer het veldobject en verander de naam van het object in *lijstVeld*.
3. Inspecteer het object *lijstVeld* en verander het kenmerk 'Weergavetype' in 'Afrol-bewerken'. Er verschijnt een dialoogvenster. Voer geen waarden in. Kies alleen 'OK'.
4. Kies 'Formulier | Objectenschema'. Zoals het Objectenschema in Afbeelding 16-8 laat zien, is een afrol-bewerkveld een samengesteld object. Het veld bestaat uit meer dan één object. Een afrol-bewerkveld is samengesteld uit het veldobject zelf, dat de veldobjectwaarde weergeeft, en het lijstobject, een knop waarop u klikt om een lijst met waarden te openen. (Het programmeren van de lijst wordt verderop in deze paragraaf uitgelegd.)

### Afbeelding 16-7 Een afrol-bewerkveld is een samengesteld object

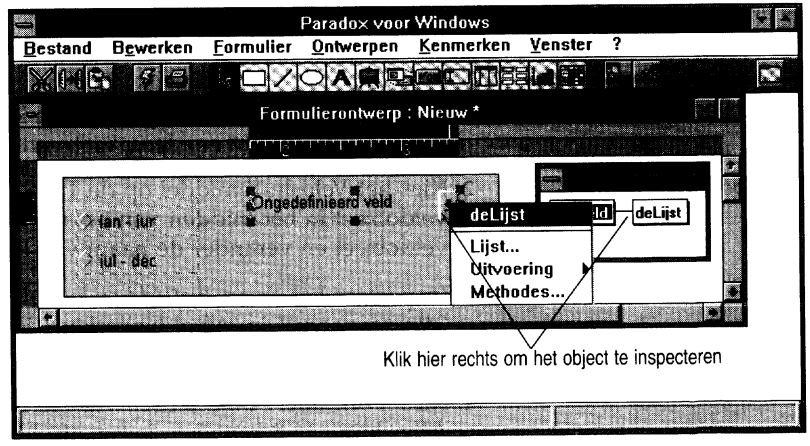
Een afrol-bewerkveld bestaat uit twee objecten: het veld zelf en de lijst




5. Klik rechts op het lijstobject in het Objectenschema om het te inspecteren, zoals in Afbeelding 16-7. Verander de naam van het lijstobject in *deLijst*. Sluit het venster met het Objectenschema.

### Afbeelding 16-8 Een object inspecteren met het Objectenschema

U kunt het Objectenschema gebruiken om een object te inspecteren. Klik rechts op de naam van het object.



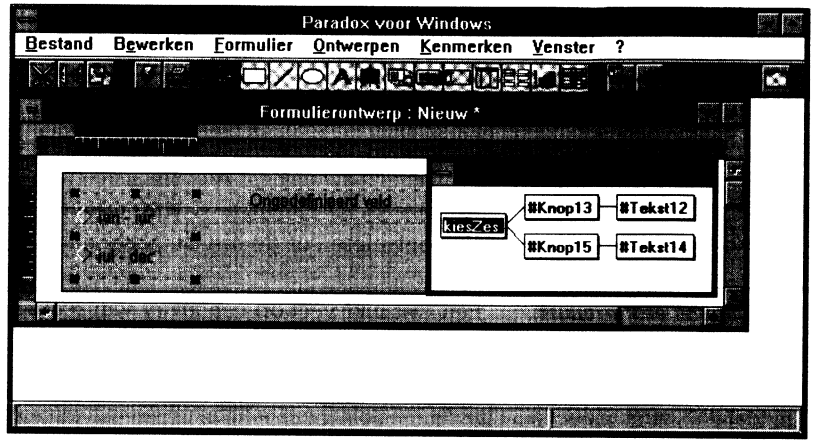

6. Plaats een ander veldobject links van *lijstVeld*. Inspecteer dit veldobject en verander de naam van dit veldobject in *kiesZes*.

7. Verander het kenmerk 'Weergavetype' van *kiesZes* in 'Keuzeknoppen'. Voer in het dialoogvenster dat verschijnt, de volgende waarden in:

**jan - jun**
**jul - dec**

8. Kies 'OK' om het dialoogvenster te sluiten.
9. Kies opnieuw 'Formulier | Objectenschema'. Zoals u ziet in Afbeelding 16-9, zijn keuzeknoppen ook samengestelde objecten.

Afbeelding 16-9 Keuzeknoppen in het Objectenschema




## Methodes koppelen

Alle code in dit voorbeeld is gekoppeld aan *kiesZes*.

1. Sluit het venster met het Objectenschema.
2. Inspecteer *kiesZes* en kies 'Methodes'.
3. Selecteer **Var**, **open**, **close** en **newValue** in het methodevenster. Kies vervolgens 'OK' om vier vensters van de ObjectPAL-Editor te openen. Bewerk de tekst in elk venster als volgt:

*Var-venster*

```
Var
   maandenTC  TCursor
   i          SmallInt
endVar
```

*open-methode*

```
method open(var eventInfo Event)
   maandenTC.open("maanden.db")
   self.value = "jan - jun"     ; wijs een beginwaarde toe
; de vorige instructie activeert de newValue-methode
endMethod
```

*close-methode*

```
method close(var eventInfo Event)
  maandenTC.close()
endMethod
```

*newValue-methode*

```
method newValue(var eventInfo ValueEvent)
if eventInfo.reason() = EditValue or eventInfo.reason() = StartUpValue then
      lijstVeld.deLijst.list.count = 0                    ; stel de lijst in
      switch
```

```
            case self = "jan - jun" : maandenTC.moveToRecord(1) ; begin met
                                                                ; januari
            case self = "jul - dec" : maandenTC.moveToRecord(7) ; begin met
                                                                ; juli
        endSwitch

        for i from 1 to 6                                       ; maak de lijst
            lijstVeld.deLijst.list.selection = i
            lijstVeld.deLijst.list.value = maandenTC.maandNaam
            maandenTC.nextRecord()
        endFor
    endIf
    endMethod
```

4. Kies 'Taal | Syntaxis controleren' om uw methodes te controleren op fouten en breng indien nodig correcties aan.
5. Kies 'Bestand | Opslaan' om uw werk op te slaan.
6. Kies 'Formulier | Gegevens tonen' om het formulier te starten.
7. Klik op de keuzeknop. Klik vervolgens op de afrollijst om te controleren of de lijst de juiste maanden bevat.

***Werking van een afrollijst***

Een lijst lijkt veel op een array van reeksen: beide indexeren de elementen en beide beginnen te tellen vanaf 1. U werkt met een afrollijst door de volgende kenmerken te manipuleren: 'Lijst.Count', 'Lijst.Selection' en 'Lijst.Value'.

- Het kenmerk 'Lijst.Count' bevat het aantal elementen in de lijst. Als u Lijst.Count = 0 opgeeft, wordt de lijst leeggemaakt.
- Het kenmerk 'Lijst.Selection' bevat de indexen van de lijst-elementen. Het eerste element in een lijst heeft Lijst.Selection = 1, het tweede element heeft Lijst.Selection = 2 enzovoort.
- Het kenmerk 'Lijst.Value' bevat de reeks van een bepaalde selectie.

In de code die is gekoppeld aan de **newValue**-methode zorgt de volgende instructie ervoor dat de code voor het maken van een lijst alleen wordt uitgevoerd als de **newValue**-methode wordt geactiveerd doordat u het formulier de eerste keer start of doordat u op een keuzeknop klikt. Als **newValue** om een andere reden wordt geactiveerd, wordt de code voor het maken van een lijst niet uitgevoerd.

```
if eventInfo.reason() = StartUpValue or eventInfo.reason() = EditValue then
```

# Zelfverhogend sleutelveld programmeren

Een veel voorkomende taak bij databaseprogrammering is het automatisch toewijzen van een unieke sleutelwaarde aan elk nieuw record. Als u bijvoorbeeld een applicatie maakt voor de invoer van

bestelinformatie, hebt u voor elke nieuwe bestelling een uniek ordernummer nodig. In een voorgaande les in dit handboek les wordt beschreven hoe u dit eenvoudig doet in een single-user applicatie. Het volgende voorbeeld presenteert een manier die u kunt gebruiken in een multi-user applicatie.

Stel dat u al een formulier hebt gemaakt voor de invoer van orderinformatie en een unieke waarde wilt genereren om elke nieuwe order aan te duiden. U declareert dan eerst een TCursor in het Var-venster van de pagina om deze beschikbaar te maken voor alle methodes van de pagina.

```
var
   ordNrTC TCursor
endVar
```

Vervolgens koppelt u de volgende code aan de ingebouwde **open**-methode van de pagina. Deze code opent een TCursor voor de tabel die het volgende ordernummer bevat, en stelt het kenmerk 'TabStop' van het veldobject *Ordernr_* in op False. Dit voorkomt dat de gebruiker de cursor naar het veldobject verplaatst en een waarde invoert.

```
method open(var eventInfo Event)
   ordNrTC.open("OrdNum.db")
   ordNrTC.edit()
   Ordernr_.TabStop = False
endMethod
```

***Tip*** Het is verstandig om sleutelwaarden in een andere tabel op te slaan dan de tabel waarin u gegevens bewerkt. In dit voorbeeld vertegenwoordigen de sleutelwaarden ordernummers en worden deze opgeslagen in de tabel *Ordnum*. Zo kunt u de tabel *Ordnum* vergrendelen om een uniek ordernummer te krijgen, zonder de hele tabel *Order* te vergrendelen, wat zou voorkomen dat andere gebruikers in die tabel kunnen werken.

Ten slotte moet u code koppelen aan de ingebouwde **action**-methode van de pagina. Deze code probeert de tabel te vergrendelen die het volgende ordernummer bevat. Als dit lukt, wijst de code een waarde toe aan het veldobject *Ordernr_* op het formulier en wordt de waarde in de tabel verhoogd. Als het niet lukt, wordt er een dialoogvenster geopend om de gebruiker te informeren.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataInsertRecord then
      doDefault
      if OrdNrTC.lockRecord() then
         Ordernr_.value = OrdNrTC.Ordernum + 1
         OrdNrTC.OrderNum = Ordernr_.value
         OrdNrTC.unlockRecord()
      else
         msgStop("Probleem", "Kan de tabel OrdNum niet vergrendelen")
      endif
   endIf
endMethod
```

Raadpleeg Hoofdstuk 12 voor meer informatie over het maken van multi-user applicaties.

HOOFDSTUK 17

# Systeemgegevens-objecten

Als u ObjectPAL-applicaties ontwikkelt, hebt u soms informatie nodig over de Windows-omgeving van een gebruiker, over de instellingen van het werkstation, over bestandsdirectories en netwerken. Met de systeemgegevens-objecten kunt u deze informatie opvragen en ermee werken. De systeemgegevens-objecten onderscheiden zich van gegevensmodel-objecten, omdat systeemgegevens-objecten geen gegevens in tabellen opslaan. In dit hoofdstuk worden deze objecten, waarvan in Tabel 17-1 een overzicht wordt gegeven, besproken en wordt het gebruik ervan gedemonstreerd.

Tabel 17-1 Systeemgegevens-objecten

| Type | Beschrijving |
|---|---|
| DDE | Een Dynamic Data Exchange-koppeling tussen Paradox en een andere applicatie |
| FileSystem | Verschaft toegang tot schijfbestanden |
| Library | Een verzameling eigen code |
| Session | Informatie over applicaties en gebruikersaantallen |
| System | Informatie over het computersysteem waarop de applicatie actief is |
| TextStream | Tekst die wordt geschreven naar en gelezen uit schijfbestanden |

## DDE: koppeling met andere applicaties

Dynamic data exchange (DDE) is een Windows-protocol waarmee Paradox gegevens gezamenlijk kan gebruiken met andere applicaties die zich overeenkomstig het DDE-protocol gedragen. Als u DDE-methodes gebruikt, kunt u Paradox-gegevens maken en opslaan in een andere applicatie. U kunt DDE-methodes ook gebruiken om opdrachten en Paradox-gegevens naar de andere applicatie te sturen.

***Opmerking*** ObjectPAL en Paradox ondersteunen OLE (Object Linking and Embedding), een ander protocol voor gezamenlijk gebruik van

gegevens. Omdat OLE-gegevens in een tabel kunnen worden opgeslagen, is het OLE-type ingedeeld bij de gegevenstypes (zie Hoofdstuk 15).

## DDE-conversaties

Het gezamenlijk gebruik van gegevens via een DDE-koppeling lijkt op een conversatie. Een DDE-conversatie verloopt in drie stappen:

- Open de DDE-conversatie.
- Converseer.
- Sluit de DDE-conversatie.

De aard van een DDE-conversatie is afhankelijk van de applicatie waarmee u converseert. Raadpleeg de DDE-documentatie van de applicatie voor informatie over gegevensuitwisseling, bijvoorbeeld over de opdrachten die de applicatie accepteert en de manier waarop fouten worden behandeld.

### DDE-conversatie openen

Met **open** geeft u de volgende zaken op:

- De applicatie waarmee u converseert. De applicatie moet zich in het pad van de gebruiker bevinden. (Raadpleeg de documentatie van DOS en Windows voor informatie over het instellen van een pad.)
- Het onderwerp van de conversatie.
- Een element binnen dat onderwerp (optioneel).

De volgende methode opent bijvoorbeeld een conversatie met ObjectVision (VISION), dat zich in uw pad moet bevinden. Geef geen bestandsnaamextensie voor de applicatie op. Het onderwerp is een formulier met de naam *Adres* en het element wordt gevormd door de gegevens in het veld *Achternaam*. De dubbele backslash-tekens zijn vereist als u een pad opgeeft.

```
var ddeKoppeling DDE endVar
ddeKoppeling.open("vision", "C:\\vision\\formlrn\\Adres.ovd", "Achternaam")
```

Als u DDE-elementen in combinatie met **open** gebruikt, zijn twee aspecten van belang: de elementen zijn optioneel en verschillen van applicatie tot applicatie.

*Elementen zijn optioneel*

Als u het ObjectVision-formulier in het vorige voorbeeld wilt openen zonder een specifiek veld te noemen, kunt u het *element* weglaten uit de **open**-instructie, bijvoorbeeld:

```
var ddeKoppeling DDE endVar
ddeKoppeling.open("vision", "C:\\vision\\formlrn\\Adres.ovd")
```

Nadat u de koppeling hebt geopend, kunt u **setItem** (zie verderop) gebruiken om een element op te geven.

*Elementen verschillen van applicatie tot applicatie*

Er bestaat niet één enkele syntaxis om elementen in een DDE-conversatie op te geven, omdat verschillende applicaties op een andere manier gegevens behandelen. Rijen en kolommen zijn bijvoorbeeld in een spreadsheet van grote betekenis, maar niet in een tekenprogramma. Raadpleeg de documentatie van een applicatie om te bepalen hoe u een DDE-onderwerp moet opgeven.

## Conversatie in DDE-stijl

Zodra de conversatie tot stand is gekomen, kunt u de koppeling gebruiken om gegevens uit de andere applicatie op te vragen. De elementwaarde wordt *niet* in de DDE-variabele opgeslagen. U gebruikt de DDE-variabele om de huidige gegevens van het element op te vragen of nieuwe gegevens naar het element te sturen. Zolang de DDE-koppeling bestaat, kunt u de DDE-variabele gebruiken om gegevens op te vragen en te versturen.

*Gegevens opvragen*

Als u met een DDE-koppeling gegevens wilt opvragen, wijst u de waarde van de DDE-variabele aan een andere variabele toe. De variabele moet van het type AnyType zijn of van een type dat door AnyType wordt vertegenwoordigd. Als u niet zeker bent van het gegevenstype van de doelgegevens of als het gegevenstype kan variëren, gebruikt u een AnyType-variabele. In het volgende voorbeeld wordt ervan uitgegaan dat de gegevens in het veld 'Achternaam' een tekenreeks vormen en wordt de waarde van de DDE-koppeling toegewezen aan de String-variabele *koppelNaam*.

```
var
   mijnKoppeling DDE
   koppelNaam String
endVar
mijnkoppeling.open("vision", "C:\\vision\\formlrn\\Addres.ovd", "Achternaam")
                             ; element is veld 'Achternaam'
koppelNaam = mijnKoppeling ; stelt koppelNaam in op waarde van veld
                             ; 'Achternaam' van het OV-formulier Adres
```

U kunt op twee manieren meerdere waarden opvragen van een applicatie:

- Gebruik **setItem** om het element te veranderen.
- Open meer dan één DDE-koppeling.

In het volgende voorbeeld wordt **setItem** tweemaal gebruikt om de waarden van de twee velden in een ObjectVision-formulier op te vragen.

```
var
   krijgNamen DDE
   voorNaam, achterNaam AnyType
endVar

; koppeling met het ObjectVision-formulier
krijgNamen.open("vision", "C:\\vision\\formlrn\\Adres.ovd")

krijgNamen.setItem("Achternaam")              ; element is veld 'Achternaam'
achterNaam = krijgNamen                       ; stelt achterNaam in op veld
```

```
                                                    ; 'Achternaam'

krijgNamen.setItem("Voornaam")                      ; element is veld 'Voornaam'
voorNaam = krijgNamen                               ; stelt voorNaam in op veld
                                                    ; 'Voornaam'

msgInfo("De naam is:", voorNaam + space(1) + achterNaam)
krijgNamen.close()                                  ; sluit de koppeling
```

Het volgende voorbeeld opent twee DDE-koppelingen en slaat de waarden op in ObjectPAL-variabelen.

```
var
   d1, d2 DDE
   voorNaam, achterNaam AnyType
endVar

d1.open("vision", "C:\\vision\\formlrn\\Adres.ovd", "Voornaam")
d2.open("vision", "C:\\vision\\formlrn\\Adres.ovd", "Achternaam")

voorNaam = d1
achterNaam = d2

msgInfo("De naam is:", voorNaam + space(1) + achterNaam)
d1.close()
d2.close()
```

Het volgende voorbeeld opent een DDE-koppeling met Quattro Pro voor Windows van Borland, haalt een waarde uit een cel in de spreadsheet en stelt op basis van die waarde de waarde van een andere cel in.

```
method pushButton(var eventInfo Event)
var
   quattroKoppeling DDE
   koppelWaarde AnyType
endVar

quattroKoppeling.open("qpw", "C:\\qpw\\werkbk1.wb1", "A:A1")
koppelWaarde = quattroKoppeling

if SmallInt(koppelWaarde) < 100 then  ; zet koppelWaarde om naar een SmallInt
   quattroKoppeling.setItem("B:B1")   ; ga naar pagina B, cel B1
   quattroKoppeling = 999             ; stel celwaarde in op 999
else
   quattroKoppeling.setItem("C:A2")   ; ga naar pagina C, cel A2
   quattroKoppeling = 101             ; stel celwaarde in op 101
endIf

quattroKoppeling.close()

endmethod
```

*Gegevens versturen*

U stuurt gegevens naar een andere applicatie door een waarde aan de DDE-variabele toe te wijzen. De volgende instructies openen bijvoorbeeld een DDE-koppeling met het ObjectVision-formulier ADRES.OVD en stellen vervolgens de waarde van de velden 'Voornaam' en 'Achternaam' in.

```
var
   ddeVar DDE
endVar
```

```
ddeVar.open("vision", "C:\\vision\\formlrn\\Adres.ovd")
ddeVar.setItem("Voornaam")

ddeVar = "Frank" ; stel de waarde van het veld 'Voornaam' in het OV-formulier
                 ; in op Frank

ddeVar.setItem("Achternaam")
ddeVar = "Borland" ; stel de waarde van het veld Achternaam in het OV-formulier
                   ; in op Borland
```

*Opdrachten versturen*

U kunt **execute** gebruiken om opdrachten naar de andere applicatie te sturen. De aard van deze opdrachten verschilt van applicatie tot applicatie. Het volgende voorbeeld gebruikt de ObjectVision-functie @SETTITLE om de tekst op te geven die op de titelbalk van ObjectVision moet worden weergegeven.

```
var dl DDE endVar
; open een koppeling met het ObjectVision-formulier Adres
dl.open("vision", "C:\\vision\\formlrn\\Adres.ovd")
dl.execute("[@SETTITLE(\"Ik ben de baas!\")]")
```

### DDE-conversatie sluiten

Als u een DDE-conversatie wilt sluiten, gebruikt u **close**. Deze methode sluit de DDE-koppeling van Paradox met de andere applicatie, maar de andere applicatie blijft wel open. Als u de andere applicatie wilt sluiten, kunt u **execute** in combinatie met de applicatie-specifieke opdracht (indien beschikbaar) gebruiken voordat u **close** gebruikt. Het volgende voorbeeld vraagt gegevens op van een ObjectVision-formulier en gebruikt vervolgens de ObjectVision-functie @APPEXIT om ObjectVision te sluiten.

```
var
   dl DDE
   voorNaam AnyType
endVar

; open een koppeling met het ObjectVision-formulier Adres
dl.open("vision", "C:\\vision\\formlrn\\Adres.ovd", "Voornaam")
voorNaam = dl
msgInfo("Voornaam", voorNaam)

dl.execute("[@APPEXIT]") ; @APPEXIT is een ObjectVision-functie
dl.close()
```

## FileSystem: toegang tot schijfbestanden en directories

Het FileSystem-type biedt methodes voor het werken met schijfbestanden, stations en directories. Een FileSystem-variabele zorgt voor een handle, een variabele die u in ObjectPAL-instructies kunt gebruiken om te werken met een directory of een opgegeven groep bestanden in een directory. In Tabel 17-2 wordt een overzicht gegeven van enkele taken en van de methodes om deze taken uit te voeren.

Tabel 17-2 Veel gebruikte taken en methodes van FileSystem

| Taak | Methodes |
|---|---|
| Werken met bestanden | accessRights, copy, delete, findFirst, findNext, fullName, name, rename, size |
| Werken met stations | drives, existDrive, freeDiskSpace, getDrive, isFixed, isRemote, isRemovable, setDrive, totalDiskSpace |
| Werken met directories | deleteDir, getDir, makeDir, setDir, windowsDir, windowsSystemDir |

***Opmerking*** Wees voorzichtig als u ObjectPAL en FileSystem-methodes gebruikt om met tabellen te werken. Het is namelijk mogelijk dat er bestanden aan de tabellen zijn verbonden. Stel dat de tabel *Klant* het sleutelveld 'Klantnr.' heeft en een memoveld bevat. In dit geval hebt u drie verbonden bestanden: het tabelbestand, KLANT.DB, het indexbestand, KLANT.PX, en een bestand met de memogegevens, KLANT.MB. Voor bepaalde bewerkingen (bijvoorbeeld voor het kopiëren van een tabel), is het van belang alle verbonden bestanden mee te nemen. Raadpleeg het *Handboek* voor meer informatie.

## Werken met een FileSystem-variabele

In veel gevallen bestaat de eerste stap bij het werken met een FileSystem-variabele uit het gebruik van **findFirst** om te bepalen of de directory bestanden bevat. U kunt deze stap het best beschouwen als de initialisatie van de FileSystem-variabele.

Deze methode initialiseert de FileSystem-variabele *c*. Vervolgens wordt *c* gebruikt als een handle om te bepalen of de directory C:\WINDOWS bestanden bevat. Als er bestanden aanwezig zijn, geeft deze methode informatie over uw toegangsrechten tot deze directory; anders wordt gemeld dat er geen bestanden zijn gevonden, zoals in het volgende voorbeeld:

```
var c FileSystem endVar
if c.findFirst("C:\\windows\\*.*") then ; let op dubbele backslash-teken
   msgInfo("Toegangsrechten", c.accessRights())
else
   msgInfo("Windows-directory", "Geen bestanden gevonden.")
endIf
```

Als u met een FileSystem-object werkt, plaatst u de directorypaden tussen aanhalingstekens en gebruikt u twee backslash-tekens. U kunt hoofdletters en kleine letters door elkaar gebruiken. Stel dat het DOS-pad waarmee u werkt, als volgt luidt:

```
C:\WINDOWS\SYSTEM
```

Als u dat pad in een methode wilt opgeven, gebruikt u:

```
"c:\\windows\\system"
```

U kunt de jokertekens * en ? met FileSystem-objecten gebruiken, zoals u deze ook in DOS en Windows gebruikt. De volgende code vindt bijvoorbeeld bestanden met de naam ORDER.DB en VERKOOP.DBF.

```
c.findFirst("C:\\tabellen\\*.db?")
```

***Belangrijk*** U kunt ObjectPAL niet gebruiken om de werkdirectory (:WORK:) of de privé-directory (:PRIV:) van een actieve applicatie te veranderen. Als u :WORK: of :PRIV: verandert, sluit Paradox namelijk alle geopende vensters, inclusief het venster met de instructie die directories verandert.

In plaats daarvan kunt u de System-procedure **execute** gebruiken om Paradox nog een keer te starten en opdrachtregel-schakelopties gebruiken om :WORK:, :PRIV: en een opstartdocument op te geven, zoals in het volgende voorbeeld wordt getoond.

De volgende code start Paradox nog een keer, stelt :WORK: in op C:\PDOXWIN\FORM, stelt :PRIV: in op C:\PDOXWIN\PRIVE en start het formulier ORDER.FSL.

```
method pushButton(var eventInfo Event)
   execute("pdoxwin.exe -w c:\\pdoxwin\\form -p c:\\pdoxwin\\prive order.fsl")
endmethod
```

Raadpleeg *Aan de slag* voor meer informatie over configureren via de opdrachtregel.

Naast de FileSystem-methodes waarvan in Tabel 17-2 een overzicht wordt gegeven, kent ObjectPAL twee andere methodes: de **enumFileList**-methode van het FileSystem-type en de **fileBrowser**-methode van het System-type.

### *Bestandsinformatie in een tabel weergeven*

U gebruikt **enumFileList** om bestandsinformatie in een Paradox-tabel of een array te plaatsen. Vervolgens kunt u de ObjectPAL-methodes voor het manipuleren van tabellen gebruiken om met de gegevens te werken. De volgende code zoekt bijvoorbeeld in de directory C:\WINDOWS naar bestanden met de extensie .EXE en schrijft informatie over deze bestanden naar een Paradox-tabel met de naam WINAPPS.DB.

```
if c.findFirst("C:\\windows\\*.exe") then
   c.enumFileList("C:\\windows\\*.exe", "winapps.db")
endIf
```

Nu kunt u de tabel doorzoeken, een query uitvoeren, de gegevens bewerken enzovoort.

Het volgende voorbeeld zoekt in de opgegeven directory naar bestandsnamen die Paradox-formulieren vertegenwoordigen, slaat de namen in een array op en geeft de namen vervolgens in een pop-up menu weer.

```
method pushButton(var eventInfo Event)

   var
      fs FileSystem
      formDir String
      formNamen Array[] String
      p PopUpMenu
   endvar

   formDir = "C:\\pdoxwin\\formlrn\\*.f?l"

   if fs.findFirst(formDir) then
      fs.enumFileList(formDir, formNamen)
      p.addArray(formNamen)
      hetFormulier = p.show() ; geeft een pop-up menu met formuliernamen weer
   endIf
endmethod
```

## *fileBrowser*

U kunt de **fileBrowser**-procedure van het System-type gebruiken om de gebruiker de mogelijkheid te geven een of meer bestanden te kiezen met behulp van de ingebouwde bladermodus van Paradox. U kunt de keuze(n) van de gebruiker opslaan in een String of een Array, zoals in het volgende voorbeeld.

```
var
   eenBestand String
   veelBestanden Array[] String
   tView TableView
endVar
fileBrowser(eenBestand) ; Geeft de bladermodus weer en wacht
                        ; tot de gebruiker één bestand kiest.
                        ; Variabele eenBestand slaat de gekozen bestandsnaam op.
if isTable(eenBestand) then
   tView.open(eenBestand)
endIf

; de gebruiker kan meerdere bestanden selecteren en de bestandsnamen worden in
; een array opgeslagen
fileBrowser(veelBestanden)

veelBestanden.view() ; geeft de keuzes van de gebruiker weer
```

U kunt een record ook als een argument aan **fileBrowser** doorgeven om op te geven welke gegevens de bladermodus moet weergeven. U kunt er bijvoorbeeld voor zorgen dat de bladermodus alleen Paradox-tabellen weergeeft of alleen formulieren of formulieren en rapporten.

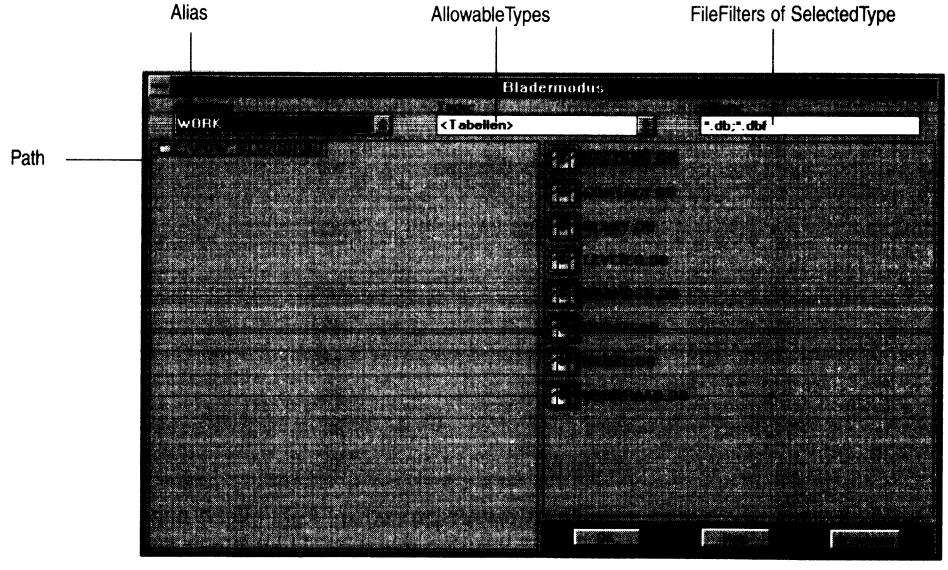
ObjectPAL kent een speciaal gegevenstype, FileBrowserInfo, dat u *alleen* met de **fileBrowser**-procedure kunt gebruiken. FileBrowserInfo is een Record met de volgende structuur:

```
TYPE FileBrowserInfo =
  Record
    x, y, w, h   SmallInt      ; Grootte van bladermodusvenster in twips
    WindowStyle  LongInt       ; Vensterstijlconstanten.
    AllowableTypes LongInt     ; Zie Tabel 17-3
    SelectedType   LongInt     ; Een van de AllowableTypes
    FileFilters    String      ; De bestandsspecificatie in invoervak
    Alias          String      ; Alias of stationsnaam
    Path           String      ; Pad ten opzichte van ALIAS
```

```
  endRecord
ENDTYPE
```

Deze recordstructuur is voorgedefinieerd en in ObjectPAL ingebouwd. U hoeft dus alleen een variabele van het type FileBrowserInfo te declareren en waarden aan de velden toe te wijzen, zoals in het voorbeeld aan het einde van deze paragraaf. U hoeft het type niet elke keer zelf te declareren als u het wilt gebruiken. In Afbeelding 17-1 ziet u de bladermodus en de gebieden die door de verschillende velden van het FileBrowserInfo-record worden beïnvloed.

Afbeelding 17-1 Bladermodus-gebieden die worden beïnvloed door FileBrowserInfo




Nadat **fileBrowser** is aangeroepen, worden in de velden 'Alias', 'Path' en 'FileFilter' de waarden ingevuld uit het dialoogvenster 'Bladermodus'. Met andere woorden, u kunt vaststellen wat de gebruiker in die gebieden van de bladermodus heeft ingevoerd.

Het veld 'AllowableTypes' geeft aan wat in de bladermodus in de afrol-bewerklijst van het paneel 'Type' verschijnt. Het veld 'SelectedType' geeft aan welke van de AllowableTypes op het moment is geselecteerd. In Tabel 17-3 wordt een overzicht gegeven van geldige ObjectPAL-constanten die in de velden 'SelectedType' en 'AllowableTypes' kunnen worden gebruikt.

Tabel 17-3 Constanten voor de velden 'AllowableTypes' en 'SelectedType'

| Constante | Extensie | Beschrijving |
|---|---|---|
| fbFiles | *.* | Alle bestanden |
| fbTable | *.db | Paradox-tabellen |
| fbQuery | *.qbe | QBE-bestanden |
| fbForm | *.fsl, *.fdl | Paradox-formulieren |
| fbReport | *.rsl, *.rdl | Paradox-rapporten |
| fbScript | *.ssl, *.sdl | Paradox-scripts |
| fbGraphic | *.bmp, *.pcx, *.tif, *.gif, *.eps | Verschillende grafische bestandsformaten |
| fbBitmap | *.bmp | Bitmap-afbeeldingen van Windows |
| fbText | *.txt | Tekstbestanden |
| fbAllTables | *.db, *.dbf | Gebruikers- en systeemtabellen |
| fbTableView | *.tv | Tabelweergavebestanden |
| fbParadox | *.db | Paradox-tabellen |
| fbDBase | *.dbf | dBASE-tabellen |
| fbASCII | *.txt | Tekstbestanden |
| fbQuattroProWindows | *.wt1 | Werkbladen van Quattro Pro voor Windows |
| fbQuattroPro | *.wq1 | Werkbladen van Quattro Pro voor DOS |
| fbQuattro | *.wkq | Quattro-werkbladen |
| fbLotus2 | *.wk1 | Lotus-werkbladen (versie 2) |
| fbLotus1 | *.wks | Lotus-werkbladen (versie 1) |
| fbExcel | *.xls | Excel-werkbladen |
| fbIni | *.ini | .INI-bestanden |
| fbLibrary | *.lsl | Paradox-bibliotheken |

De **fileBrowser**-procedure kijkt alleen naar de namen die in de structuur worden gegeven. U kunt een andere recordstructuur aan deze procedure doorgeven. De velden met de juiste namen worden dan gevonden en gebruikt. Met andere woorden, u kunt een eenvoudigere recordstructuur opgeven die alleen de elementen bevat die voor u van belang zijn.

*FileBrowserInfo*

Koppel de volgende code aan de ingebouwde **pushButton**-methode van een knop. Als de methode wordt uitgevoerd, wordt de bladermodus aangeroepen en wordt gewacht totdat u een bestand kiest. Vervolgens wordt informatie over uw keuze weergegeven in het statusgebied.

```
method pushButton(var eventInfo Event)

var
   fbi FileBrowserInfo ; Declareer een variabele die de voorgedefinieerde
```

```
                              ; FileBrowserInfo-recordstructuur gebruikt
   selectBestand String
endVar

; De volgende instructies wijzen waarden toe aan velden in het
; record van FileBrowserInfo
fbi.Alias = "WERK" ; Zoek de huidige werkdirectory
fbi.AllowableTypes = fbTable + fbForm ; Zoek naar tabellen en formulieren

; Geef de bladermodus weer en verwerk de selectie van de gebruiker
if fileBrowser(selectBestand, fbi) then
   message("U selecteerde ", selectBestand," met het pad ", fbi.path)
else
   message("U selecteerde Annuleren")
endif

endMethod
```

# Library: een verzameling eigen code

Een bibliotheek is een bestand waarin eigen methodes, eigen procedures, variabelen, constanten, en door de gebruiker gedefinieerde gegevenstypes worden opgeslagen. Als u bibliotheken gebruikt, kunt u vaak gebruikte routines opslaan en onderhouden en code gezamenlijk laten gebruiken door verschillende formulieren.

***Belangrijk*** Formulieren hebben geen directe toegang tot variabelen die in een bibliotheek zijn gedeclareerd. U moet eigen methodes of procedures schrijven die de waarden instellen en opvragen.

Het werken met een bibliotheek lijkt in veel opzichten op het werken met een formulier. Als u bijvoorbeeld een formulier wilt maken, kiest u 'Bestand | Nieuw | Formulier' en als u een bibliotheek wilt maken, kiest u 'Bestand | Nieuw | Bibliotheek'. Net als een formulier heeft een bibliotheek ingebouwde methodes. Net als aan een formulier, voegt u code aan een bibliotheek toe met behulp van het methodevenster en de ObjectPAL-Editor. Net als bij een formulier kunt u Editor-vensters openen om eigen methodes, procedures, variabelen, constanten, gegevenstypes en externe routines te declareren.

Er zijn echter belangrijke verschillen:

- Een bibliotheek verschijnt niet in een venster tijdens de uitvoering.
- Een bibliotheek kan geen ontwerpobjecten bevatten, maar alleen code.
- In een bibliotheekmethode verwijzen instructies die *Self* gebruiken, niet naar de bibliotheek, maar naar het object dat de methode heeft aangeroepen. Zo verwijzen instructies die *Container* gebruiken, ook naar het object dat het object insluit dat de bibliotheekmethode heeft aangeroepen. Dezelfde principes gelden voor de andere objectvariabelen: *active, lastMouseClicked,*

*lastMouseRightClicked* en *subject*. Raadpleeg de online ObjectPAL Help voor meer informatie over ingebouwde objectvariabelen.

- De bereikregels zijn anders voor bibliotheken. (Zie "Bereik van een bibliotheek bepalen", verderop in dit hoofdstuk, voor meer informatie.)

In de volgende paragrafen wordt uitgelegd hoe u bibliotheken gebruikt. De volgende onderwerpen komen aan de orde:

- Bibliotheken maken
- Code aan een bibliotheek toevoegen
- Methodes uit een bibliotheek aanroepen
- Methodes van het Library-type gebruiken
- Bereik van een bibliotheek bepalen

## Bibliotheken maken

Als u een nieuwe bibliotheek wilt maken, kiest u 'Bestand | Nieuw | Bibliotheek'. Als u een bestaande bibliotheek wilt bewerken, kiest u 'Bestand | Openen | Bibliotheek'. Koppel vervolgens uw code, zoals in de volgende paragrafen wordt uitgelegd. Als uw code klaar is, kiest u 'Bestand | Opslaan' om de bibliotheek—zowel de broncode als de uitvoerbare code—op schijf op te slaan. Als u alleen de uitvoerbare code wilt opslaan, kiest u 'Taal | Aanmaken' in een Editor-venster. Zie Hoofdstuk 21 voor meer informatie over het opslaan en aanmaken van bibliotheken.

Als u een bibliotheek opslaat, geeft u een bestandsnaam op. Paradox voegt dan de extensie .LSL toe. Als u een bibliotheek aanmaakt, geeft u ook een bestandsnaam op. Paradox voegt dan de extensie .LDL toe.

Als u een nieuwe bibliotheek maakt, wordt een Editor-venster geopend dat de bibliotheek vertegenwoordigt. *U kunt geen objecten in dit venster plaatsen.* In plaats daarvan koppelt u met het methodevenster code aan de bibliotheek. Als u het methodevenster wilt openen, klikt u rechts in het venster of drukt u op *Ctrl-Spatiebalk*. Typ een naam voor de nieuwe eigen methode, net als u dat bij andere objecten doet, en kies 'OK' om een ander Editor-venster te openen.

Een bibliotheek wordt in een apart bestand op schijf opgeslagen, zodat u een bibliotheek gemakkelijk kunt kopiëren en distribueren met uw applicaties.

## Code aan een bibliotheek toevoegen

Met het methodevenster en de vensters van de ObjectPAL-Editor, kunt u op de volgende manieren code aan een bibliotheek toevoegen:

- Code koppelen aan de ingebouwde methodes
- Eigen methodes toevoegen

- ❒ Eigen procedures toevoegen
- ❒ Variabelen, constanten, gegevenstypes en externe routines declareren

### Code aan de ingebouwde methodes koppelen

Elke bibliotheek heeft de volgende ingebouwde methodes: **open**, **close** en **error**.

U kunt op dezelfde manier code aan deze ingebouwde methodes koppelen als u dat bij andere objecten doet.

Een ingebouwde **open**-methode van een bibliotheek wordt aangeroepen als de bibliotheek voor de eerste keer wordt geopend. **close** wordt aangeroepen als de bibliotheek wordt gesloten en **error** wordt aangeroepen als code in de bibliotheek een fout genereert. U gebruikt **open** om globale bibliotheekvariabelen te initialiseren en **close** om alles "op te ruimen" nadat de bibliotheek is gebruikt. De **error**-methode van een bibliotheek roept standaard de **error**-methode aan van het formulier dat de bibliotheekroutine heeft aangeroepen.

### Eigen methodes toevoegen

In deze paragraaf wordt beschreven hoe u eigen methodes aan een bibliotheek toevoegt. De eigen methodes in een bibliotheek kunnen worden aangeroepen door andere methodes in dezelfde bibliotheek, door methodes in andere formulieren en door methodes in objecten van andere formulieren. Deze toegankelijkheid maakt bibliotheken bijzonder handig.

De syntaxis voor de declaratie van een eigen methode luidt:

**METHOD** *methodeNaam* ([var | const] *argLijst*) [*terugType*]
    *Het hoofddeel van de methode komt hier*
**ENDMETHOD**

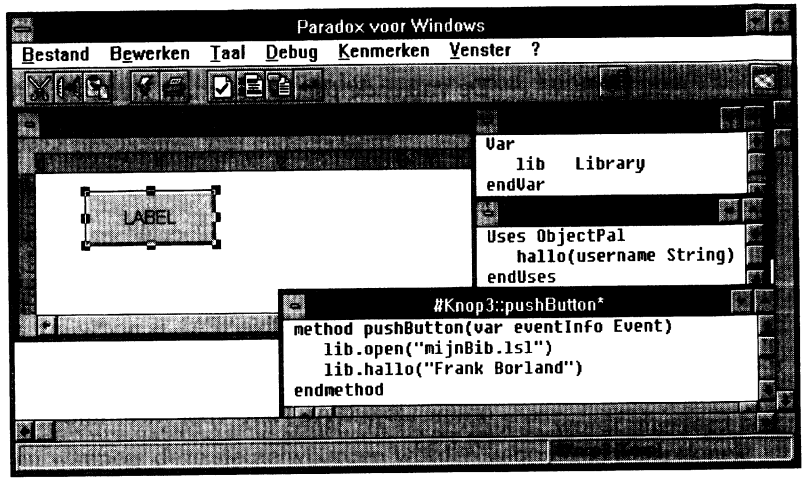
*methodeNaam* vertegenwoordigt de naam van de methode en *argLijst* is een lijst met combinaties van argumenten en gegevenstypes die door komma's worden gescheiden en optioneel worden voorafgegaan door het sleutelwoord **var** of **const**. (In Hoofdstuk 11 wordt uitgelegd hoe en wanneer deze sleutelwoorden worden gebruikt.) Het optionele argument *terugType* geeft het gegevenstype op van de waarde die door de methode wordt teruggegeven (als de methode een waarde teruggeeft).

De volgende code declareert bijvoorbeeld de eigen methode **hallo** die één argument gebruikt, namelijk een String-variabele met de naam *gebrNaam*. Deze eigen methode opent een dialoogvenster.

```
method halloGebruiker(gebrNaam String)
   msgInfo(gebrNaam, "Hallo")
endMethod
```

Zie de beschrijving van **method** in de online ObjectPAL Help voor informatie over de declaratie van methodes.

Afbeelding 17-2 Voorbeeld van een aanroep van een eigen methode vanuit een bibliotheek




In deze afbeelding opent de **pushButton**-methode een bibliotheek. Deze methode roept een eigen methode aan. Hiervoor moet eerst een Library-variabele (Var-venster) worden gedeclareerd. Ook wordt de methode gedeclareerd die moet worden aangeroepen (Uses-venster).

***Belangrijk*** De code in een bibliotheek wordt uitgevoerd namens het object dat die bibliotheek heeft aangeroepen en het object bepaalt de context voor een bibliotheekmethode die de code aanroept. Als een kader dus een bibliotheekmethode aanroept, verwijzen instructies die *Self* gebruiken naar het kader. De instructies die *Container* gebruiken, verwijzen naar het object dat het kader insluit. Hetzelfde principe geldt voor de andere ingebouwde objectvariabelen: *active*, *lastMouseClicked*, *lastMouseRightClicked* en *subject*. Raadpleeg de online ObjectPAL Help voor meer informatie over ingebouwde objectvariabelen.

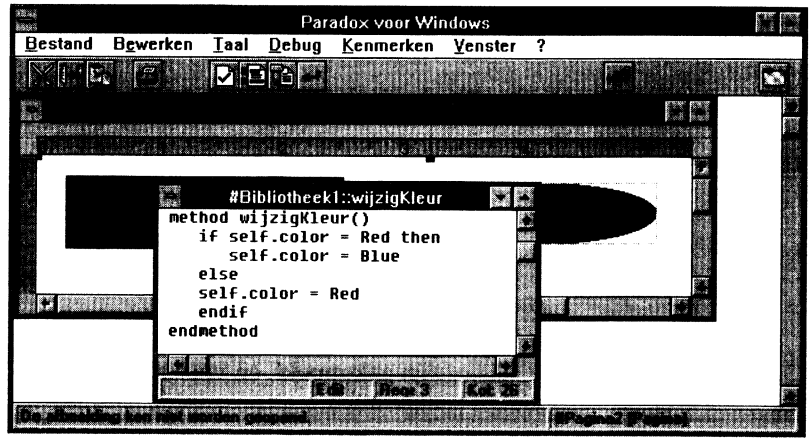
In Afbeelding 17-3 wordt bijvoorbeeld een formulier getoond dat één kader bevat met de naam *hetKader* en één ellips met de naam *deEllips*. Het formulier opent een bibliotheek die een eigen methode bevat met de naam **wijzigKleur**.

```
method wijzigKleur()
   if self.color = Red then
      self.color = Blue
   else
      self.color = Red
   endIf
endMethod
```

Als een methode die aan *hetKader* is gekoppeld, de volgende instructie uitvoert, verandert *hetKader* van kleur en als een methode die aan *hetVeld* is gekoppeld, de instructie uitvoert, verandert *hetVeld* van kleur.

```
bib.wijzigKleur()
```

Afbeelding 17-3 *Self* gebruiken in een bibliotheekroutine




*Self* verwijst in een bibliotheekroutine naar het object dat de routine heeft aangeroepen, net als in andere methodes of procedures. In deze afbeelding verwijst *Self* naar het kader als het kader **wijzigKleur** aanroept en naar de ellips als de ellips **wijzigKleur** aanroept.

## Bibliotheekmethodes

Het Library-type heeft de runtime methodes waarvan in Tabel 17-4 een overzicht wordt gegeven. U kunt deze methodes in uw eigen code gebruiken om een bibliotheek te manipuleren. Deze code kan aan een willekeurig object zijn gekoppeld, zelfs aan een andere bibliotheek. Raadpleeg de online ObjectPAL Help voor meer informatie over deze methodes.

Tabel 17-4 Library-methodes

| Methode | Beschrijving |
|---|---|
| open | Opent een bibliotheek en laadt deze in het systeemgeheugen, waardoor de bibliotheek beschikbaar is voor een of meer formulieren en bureaubladen (zie de paragraaf over het bepalen van het bereik verderop in dit hoofdstuk). |
| close | Verwijdert de bibliotheek uit het geheugen. U hoeft een bibliotheek niet expliciet te sluiten—de bibliotheek wordt automatisch verwijderd als alle verwijzende formulieren en bibliotheken zijn gesloten—maar als u deze methode gebruikt is de code beter te lezen en is het gemakkelijker te zien wat er moet gebeuren. |
| enumSource | Schrijft bibliotheekcode naar een Paradox-tabel. |
| enumSourceToFile | Schrijft bibliotheekcode naar een tekstbestand. |
| execMethod | Voert een opgegeven bibliotheekmethode uit. |

## Bereik van een bibliotheek bepalen

Het *bereik* van een bibliotheek verwijst naar de toegankelijkheid van de bibliotheek, dat wil zeggen, welke objecten toegang hebben tot de code van de bibliotheek. Het bereik van een bibliotheek wordt bepaald door de volgende factoren:

- Waar wordt de Library-variabele gedeclareerd?
- Hoe is de bibliotheek geopend?

### *Library-variabele declareren*

Het bereik van een Library-variabele volgt dezelfde regels als alle andere ObjectPAL-variabelen.

- Een variabele die in een methode wordt gedeclareerd, is beschikbaar zolang de methode wordt uitgevoerd.
- Een variabele die in het Var-venster van een object wordt gedeclareerd, is beschikbaar voor alle methodes die aan dat object zijn gekoppeld, en voor alle objecten die door dat object worden ingesloten.

Als u een bibliotheek dus beschikbaar wilt maken voor alle objecten in een formulier (zolang dat formulier wordt uitgevoerd), declareert u de Library-variabele in het Var-venster van het formulier en declareert u de bibliotheekroutines in het Uses-venster van het formulier.

Als u bijvoorbeeld de Library-variabele *wiskBib* wilt declareren, koppelt u de volgende code aan het Var-venster van het formulier.

```
var
   wiskBib Library
endVar
```

Als u de bibliotheekroutines die u wilt gebruiken, wilt declareren, koppelt u de code aan het Uses-venster van het formulier. De volgende code declareert bijvoorbeeld de routines *faculteit* en *rekenWaarde*.

```
Uses ObjectPAL
   faculteit(const eenWaarde SmallInt) LongInt
   rekenWaarde(const rang Number, const aantal Number) Number
endUses
```

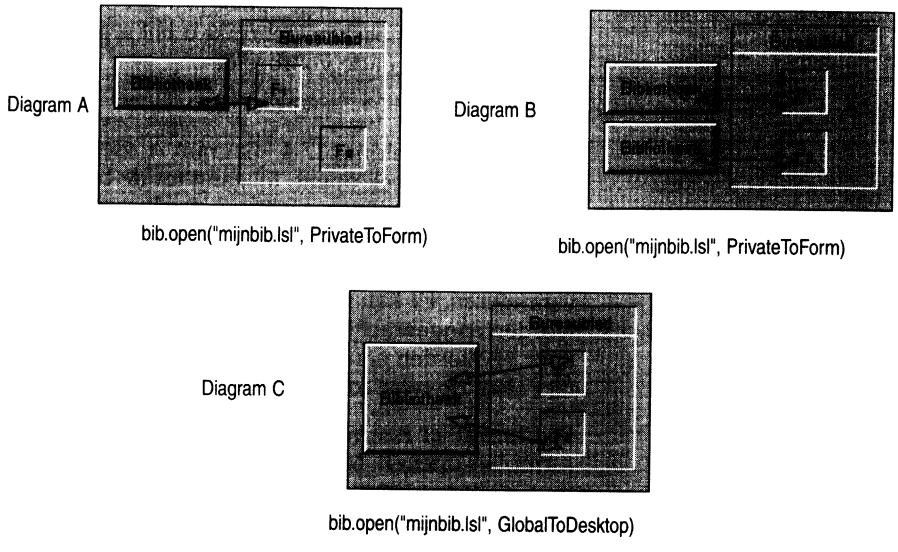
### *Bibliotheken openen*

De **open**-methode van het Library-type gebruikt argumenten die het bereik opgeven. U kunt een bibliotheek op een van de volgende manieren openen:

- Globaal voor het bureaublad: elk formulier in het bureaublad (de Paradox-sessie) kan toegang krijgen tot de bibliotheek. Dit is het standaardbereik.
- Lokaal voor het formulier: alleen het formulier dat de bibliotheek heeft geopend, heeft toegang tot de code van de bibliotheek.

In Afbeelding 17-4 worden de verschillende **open**-opties getoond.

Afbeelding 17-4 **open** gebruiken om het bereik van een bibliotheek op te geven




*Globaal voor het bureaublad*

Als u een bibliotheek wilt openen en beschikbaar wilt maken voor elk formulier in de huidige Paradox-sessie, gebruikt u het argument GlobalToDesktop. De volgende instructie opent bijvoorbeeld de bibliotheek MIJNBIB.LSL.

```
bib.open("mijnBib.lsl", GlobalToDesktop)
```

Zoals u in Afbeelding 17-4, diagram C, ziet, maakt elk formulier op het bureaublad gebruik van dezelfde geopende bibliotheek. Als het formulier F1 bijvoorbeeld een eigen methode aanroept die de waarde van een bibliotheekvariabele verandert, worden de veranderingen door F2 gezien.

***Belangrijk***

Als twee of meer formulieren dezelfde bibliotheek gezamenlijk moeten gebruiken, moet elk formulier de bibliotheek globaal voor het bureaublad openen en moet elk formulier een Uses-venster hebben dat declareert welke bibliotheekroutines worden gebruikt. In diagram C moeten bijvoorbeeld F1 en F2 de bibliotheek globaal voor het bureaublad openen en routines in een Uses-venster declareren.

Dit bereik is zeer handig in multi-formulier applicaties, omdat het op deze manier mogelijk wordt dat verschillende formulieren toegang krijgen tot dezelfde eigen methodes en dezelfde globale variabelen kunnen gebruiken.

**Opmerking** Een bibliotheek wordt standaard globaal voor het bureaublad geopend. De volgende instructies zijn dus identiek:

```
bib.open("mijnBib.lsl") ; deze instructies zijn identiek
bib.open("mijnBib.lsl", GlobalToDesktop)
```

*Lokaal voor het formulier*

Als u een bibliotheek wilt openen en het bereik tot het aanroepend formulier wilt beperken, gebruikt u het argument PrivateToForm, zoals in het volgende voorbeeld:

```
bib.open("mijnBib.lsl", PrivateToForm)
```

Zoals u in Afbeelding 17-4, diagram A, ziet, krijgt formulier F1, als het een bibliotheek opent, toegang tot de bibliotheek, maar formulier F2 krijgt dat niet. Diagram B toont echter dat F2 dezelfde bibliotheek nog een keer kan openen. Met andere woorden, F1 kan eigen methodes aanroepen die waarden van variabelen in de bibliotheek veranderen. F2 kan dat ook, maar de veranderingen die door F1 worden aangebracht, worden niet door F2 gezien. Veranderingen die door F2 worden aangebracht, worden weer niet door F1 gezien. Het lijkt alsof de twee formulieren in twee aparte bibliotheken werken.

Een formulier kan een bibliotheek slechts één keer lokaal voor het formulier openen. Met andere woorden, u kunt meerdere instructies op hetzelfde formulier plaatsen, maar elke volgende **open**-instructie sluit de huidige geopende bibliotheek en opent een nieuwe.

```
bib.open("mijnBib.lsl", PrivateToForm)
```

### Bibliotheken meerdere keren openen

Een bibliotheek kan lokaal voor het formulier worden geopend in het ene formulier en globaal voor het bureaublad in een ander formulier. Paradox laadt de bibliotheek zo nodig nog een keer.

## Library-variabelen als argumenten gebruiken

U kunt een Library-variabele als een argument gebruiken in een eigen methode of eigen procedure die aan een object in een formulier (of aan het formulier) is gekoppeld. Als u een bibliotheek als een argument doorgeeft, kunt u het gedrag van een routine (methode of procedure) veranderen, maar de onafhankelijkheid van de routine behouden. Een routine kan een bibliotheek en routines uit de bibliotheek gebruiken, maar de aanroeper kan de functie van de routines bepalen door de bibliotheek te veranderen.

De volgende eigen procedure, **berekenGetal**, kan bijvoorbeeld worden gedeclareerd in het Proc-venster van een knop. De procedure gebruikt twee argumenten: *getal*, van het type Number, en *bib*, van het type Library.

```
proc berekenGetal(var bib Library, var getal Number) Number

   ; Deze procedure is aan een knop gekoppeld,
   ; en is niet in een bibliotheek opgeslagen.
```

```
    bib.doeReken(getal)
    msgInfo("Het resultaat is:", getal)
endProc
```

Stel dat u twee bibliotheken hebt, *bibEen* en *bibTwee*. Elke bibliotheek bevat een eigen methode met de naam **doeReken**, maar de eigen methode **doeReken** in *bibEen* voert een andere berekening uit (geeft een ander resultaat terug) dan de eigen methode **doeReken** in *bibTwee*. De volgende code opent een van deze bibliotheken, afhankelijk van de waarde van de variabele *x* (de code gaat ervan uit dat de variabelen elders zijn gedeclareerd). Vervolgens wordt de Library-variabele aan de procedure **rekenGetal** doorgegeven. **rekenGetal** roept dan **doeReken** in de opgegeven bibliotheek aan en geeft in het dialoogvenster een ander resultaat weer, dat afhankelijk is van de gebruikte bibliotheek.

```
method pushButton(var eventInfo Event)
   getal = 123.45
   if x = True then
      bib.open("bibEen.lsl")
   else
      bib.open("bibTwee.lsl")
   endIf
   rekenGetal(bib, getal)
   bib.close()
endMethod
```

# Session: een kanaal naar de database-engine

Een Session-variabele vertegenwoordigt een kanaal naar de database-engine. Als u een Paradox-applicatie opent, wordt standaard één sessie geopend en kunt u ObjectPAL gebruiken om andere sessies te openen vanuit een applicatie. Het aantal sessies dat u kunt openen, varieert per omgeving, maar u kunt alleen de standaardsessie interactief met Paradox beheren. Andere sessies moet u met programmering beheren. Een Session-variabele levert een handle, een variabele die u in ObjectPAL-code kunt manipuleren om de eigenlijke Paradox-sessie te beheren.

U kunt om de volgende redenen een andere sessie willen openen:

- U wilt verschillende beveiligingsschema's gebruiken.
- U wilt verschillende aliasstructuren instellen.
- U wilt een andere behandeling van lege waarden opgeven.
- U wilt verschillende herhalingsperiodes opgeven.
- U wilt aparte vergrendelingsschema's gebruiken.

In Tabel 17-5 wordt een overzicht gegeven van enkele Session-taken en de methodes waarmee deze worden uitgevoerd.

Tabel 17-5 Taken en methodes van Session

| Taken | Methodes |
|---|---|
| Toevoegen aan een sessie | addAlias, addPassword |
| Sessie-instellingen | blankAsZero, ignoreCaseInLocate, setRetryPeriod |
| Informatie vragen over een sessie | getAliasPath, getNetUserName, retryPeriod |
| Informatie in tabellen opnemen | enumAliasNames, enumDatabaseTables, enumUsers |

***Belangrijk*** Een andere sessie openen is niet hetzelfde als Paradox nog een keer starten. Meerdere sessies werken onder hetzelfde bureaublad.

De belangrijkste attributen van een sessie zijn:

- Sessie-specifieke instellingen: elke sessie kan instellingen opgeven voor lege waarden, herhalingsperiodes, jokertekens en al dan niet onderscheid tussen hoofdletters en kleine letters bij het zoeken naar tekst.
- Wachtwoorden: elke sessie houdt een eigen lijst met wachtwoorden bij.
- Vergrendelingen: vergrendelingen worden niet gezamenlijk gebruikt door sessies. Vergrendelingen die door ObjectPAL zijn ingesteld, werken op gelijkwaardige basis samen met vergrendelingen die interactief in dezelfde sessie zijn ingesteld. Omdat ObjectPAL met Session-variabelen sessies kan maken, kunt u methodes schrijven om tabellen te vergrendelen die u niet kunt vergrendelen door Paradox interactief te gebruiken.
- Gebruikersaantal: De licentie-overeenkomsten van Paradox worden bijgehouden op basis van gebruikersaantallen die in het netwerk worden bijgehouden door de database-engine. Elke sessie telt voor één gebruiker. Dit mechanisme kan handig zijn om licenties bij te houden voor uw op Paradox gebaseerde applicatie.

## Sessies

De eerste stap bij het werken met een sessie bestaat uit het openen van de sessie. Zoals eerder is vermeld, opent Paradox standaard één sessie. Als u een handle wilt maken voor de standaardsessie, declareert u een Session-variabele en roept u de **open**-methode van het Session-type aan zonder argumenten.

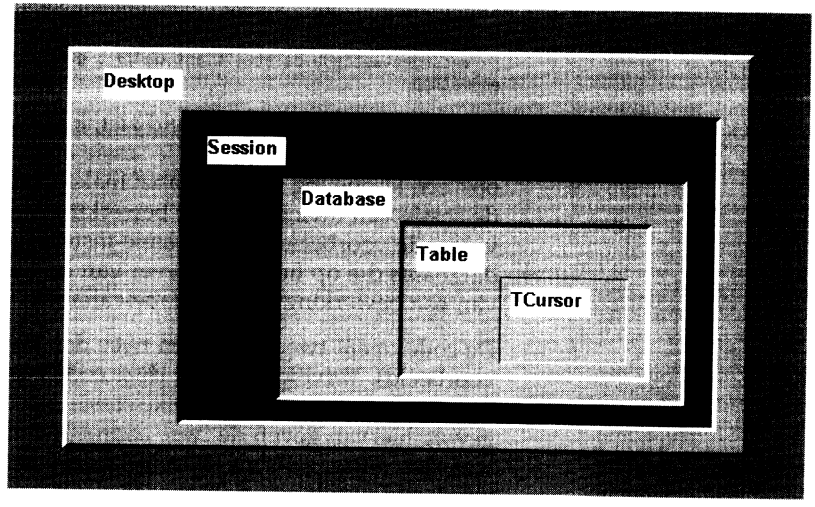
Als u een andere sessie wilt openen, declareert u een Session-variabele en roept u **open** aan met één argument, een reeks. De tekst van de reeks is niet van belang, maar u moet deze opgeven, anders krijgt u een extra handle voor de standaardsessie.

Als de volgende code wordt uitgevoerd, wordt een handle voor de standaardsessie (*standaardSes*) gemaakt en wordt een tweede sessie geopend (*tweedeSes*).

```
method pushButton(var eventInfo Event)
   var
      standaardSes, tweedeSes Session
   endVar

   standaardSes.open() ; maakt een handle voor de standaardsessie
   tweedeSes.open("Tweede sessie") ; opent een tweede sessie
endmethod
```

Als u een sessie opent, wordt een handle gemaakt waarmee u sessie-instellingen kunt opgeven. Het bureaublad houdt de instellingen voor elke sessie afzonderlijk bij. Zodra u een sessie opent, kunt u een database in die sessie openen. De sessie-instellingen zijn dan van invloed op de database. Als u een Table-variabele koppelt of een TCursor opent naar een tabel in die database, zijn de sessie-instellingen van invloed op de tabel. In Afbeelding 17-5 wordt deze hiërarchie van het bereik getoond. (Raadpleeg Hoofdstuk 16 voor informatie over databases, tabellen en TCursors.)

Afbeelding 17-5 Hiërarchie van bereiken voor Session




U kunt ObjectPAL gebruiken om meerdere sessies in hetzelfde bureaublad te openen. Het bureaublad zoekt de instellingen voor elke sessie. U kunt een database in een sessie openen en een Table-variabele koppelen of een TCursor openen voor een tabel in de database. De sessie-instellingen zijn van invloed op bewerkingen op elk niveau van de hiërarchie.

### Sessie-instellingen

Als u een sessie hebt geopend, kunt u methodes en procedures van het Session-type gebruiken om instellingen voor die sessie op te geven. Vervolgens kunt u een database in die sessie openen. De

instellingen bepalen dan hoe methodes van het Database-type, zoals **delete** en **executeQBE**, worden uitgevoerd.

Het volgende voorbeeld laat zien hoe sessie-instellingen van invloed zijn op databasebewerkingen. Stel dat de tabel *Klant* beveiligd is met "hoi" als wachtwoord. De volgende code opent twee sessies, *eersteSes* en *tweedeSes* en voegt vervolgens het wachtwoord "hoi" aan de tweede sessie toe. Daarna opent deze code een database in elke sessie. De poging om de tabel in de eerste sessie te verwijderen mislukt, omdat het wachtwoord niet aan die sessie is toegevoegd. De poging om de tabel te verwijderen lukt in de tweede sessie wel, omdat het wachtwoord is toegevoegd.

```
method pushButton(var eventInfo Event)
   var
      eersteSes, tweedeSes Session
      eersteDb, tweedeDb Database
   endVar

   eersteSes.open("Eerste sessie")   ; open eerste sessie
   tweedeSes.open("Tweede sessie")   ; open tweede sessie

   tweedeSes.addPassword("hoi")      ; voeg wachtwoord aan tweede sessie toe

   eersteDb.open(eersteSes)   ; open database in eerste sessie
   tweedeDb.open(tweedeSes)   ; open database in tweede sessie

   message(eersteDb.delete("klant.db")) ; geeft False weer
   sleep(2000)
   message(tweedeDb.delete("klant.db")) ; geeft True weer
endmethod
```

Zoals u in het vorige voorbeeld ziet, kunt u een sessie openen en vervolgens een database in die sessie openen. U kunt in die database ook een koppeling met een tabel maken, waarna de instellingen voor de sessie van invloed zijn op bewerkingen in die tabel. In het volgende voorbeeld zijn de sessie-instellingen van invloed op databases die op hun beurt weer van invloed zijn op de resultaten van een bewerking met een tabelvariabele.

De code opent twee sessies en twee databases (zoals in het vorige voorbeeld). Vervolgens roept deze code **blankAsZero** aan om op te geven hoe lege waarden in elke sessie moeten worden behandeld. Als gevolg hiervan geven de twee aanroepen van **cCount** verschillende waarden terug, zelfs als deze met dezelfde tabel werken. In dit voorbeeld wordt ervan uitgegaan dat bepaalde records in de tabel *Klant* lege waarden hebben in het veld 'Regio'. Lege waarden worden in de eerste sessie wel geteld, maar in de tweede sessie niet.

```
method pushButton(var eventInfo Event)
   var
      eersteSes, tweedeSes Session
      eersteDb, tweedeDb Database
      eersteTb, tweedeTb Table
      eersteTelling, tweedeTelling Number
   endVar
```

```
    eersteSes.open("Eerste sessie")    ; open eerste sessie
    eersteSes.blankAsZero(True)        ; geef aan hoe lege waarden moeten worden
behandeld
    eersteDb.open(eersteSes)                  ; open database in eerste sessie
    eersteTb.attach("klant.db", eersteDb)     ; maak koppeling in eerste database
    eersteTelling = eersteTb.cCount("Regio") ; tel waarden

    tweedeSes.open("Tweede sessie")
    tweedeSes.blankAsZero(False)
    tweedeDb.open(tweedeSes)
    tweedeTb.attach("klant.db", tweedeDb)
    tweedeTelling = tweedeTb.cCount("Regio")

    msgInfo("Is eersteTelling groter?", eersteTelling > tweedeTelling)
    ; geeft True weer, omdat eersteSes lege waarden telt
endMethod
```

# System: een gevarieerd type

Het System-type is een soort vergaarbak voor procedures (zoals **message**, **beep** en **sleep**) die niet in andere categorieën vallen. Het System-type omvat ook procedures (zoals **enumFonts**, **readEnvironmentString**, **readProfileString** en **sysInfo**) die informatie geven over het systeem waarop de applicatie werkt.

Daarnaast bevat dit type procedures (zoals **msgInfo** en **msgYesNoCancel**) voor de weergave van de ingebouwde dialoogvensters van ObjectPAL en **execute**, een methode die DOS-opdrachten en Windows-applicaties uitvoert.

Het volgende voorbeeld opent het klokprogramma dat bij Windows wordt geleverd (het bestand CLOCK.EXE moet zich in uw pad bevinden).

```
execute("clock.exe")
```

Het System-type omvat ook procedures die schermposities van en naar pixels en twips te converteren. De namen zijn **pixelsToTwips** en **twipsToPixels**.

## Help

ObjectPAL kent een manier om de standaard Helpapplicatie van Windows aan te roepen. De Helpapplicatie is een aparte Windows-applicatie. Als u deze wilt gebruiken, maakt u een bestand met Helpinformatie en contextreeksen en compileert u dit bestand met behulp van de Help-compiler van Microsoft (leverbaar bij Borland C++). In de documentatie die bij de Help-compiler wordt geleverd, wordt gedetailleerd uitgelegd hoe u een Helpsysteem voor Windows maakt en compileert.

Raadpleeg de voorbeeldapplicatie MAST in uw directory DUIKEN en de andere online voorbeeldapplicaties.

Zodra uw Helpsysteem is gemaakt en gecompileerd, kunt u de volgende methodes gebruiken om toegang te krijgen tot dit systeem. U vindt een overzicht van deze methodes onder "System" in de online ObjectPAL Help:

- **helpOnHelp** geeft Helpinformatie over het gebruik van de Helpapplicatie.
- **helpQuit** vertelt de Helpapplicatie dat een bepaald Helpbestand niet wordt gebruikt.
- **helpSetIndex** vertelt de Helpapplicatie welke index moet worden gebruikt.
- **helpShowContext** geeft de Helpinformatie weer die is geassocieerd met de opgegeven contextidentificatie.
- **helpShowIndex** geeft de index van een bepaald Helpbestand weer.
- **helpShowTopic** geeft de Helpinformatie weer die is geassocieerd met de opgegeven onderwerpsleutel.
- **helpShowTopicInKeywordTable** geeft de Helpinformatie weer die is geassocieerd met een sleutelwoord dat in een alternatieve sleutelwoordtabel is opgeslagen.

***Zie ook*** Zie Hoofdstuk 21 voor meer informatie over het maken van ObjectPAL-applicaties.

# TextStream: tekstbestanden

Een TextStream is een opeenvolging van tekens die wordt gelezen uit (of geschreven naar) een tekstbestand. TextStreams bevatten alleen ANSI-tekens en geen opmaakinformatie, zoals fonts, uitlijning en marges. (Als u met opgemaakte tekst wilt werken, gebruikt u Memo-objecten; zie Hoofdstuk 15.) TextStreams kunnen echter wel niet-afdrukbare tekens bevatten, zoals regelterugloop en regeldoorvoer (CR/LF).

Paradox houdt een verwijzing naar de positie in het bestand bij, die zich gedraagt als een invoegpositie in een tekstverwerkingsprogramma. De verwijzing geeft de afstand (het aantal tekens) aan tot het begin van het bestand. De telling begint bij 1 (niet bij 0, zoals in sommige andere talen).

## Werken met een TextStream

U kunt tekstbestanden maken met behulp van **create**, door Paradox interactief te gebruiken of met een tekstverwerkingsprogramma. (Dit programma een bestand als niet-opgemaakte tekst kunnen opslaan.) In Tabel 17-6 wordt een overzicht gegeven van de taken en de TextStream-methodes waarmee deze taken worden uitgevoerd.

Tabel 17-6 Taken en methodes van TextStream

| Taak | Methodes |
|---|---|
| Bestand openen en sluiten | open, close |
| Opgeven waaruit wordt gelezen en waarnaar wordt geschreven | position, setPosition |
| Lezen en schrijven | readLine, readString, writeLine, writeString |

***Opmerking*** De eerste positie in een TextStream is 1 (niet 0, zoals in sommige andere talen).

U kunt één of meer bytes, één regel tegelijk of één heel object tegelijk naar een TextStream schrijven of uit een TextStream lezen. **readLine** en **writeLine** werken met tekstregels die worden afgebakend en beëindigd door een combinatie van een regelterugloop en een regeldoorvoer (CR/LF). **readString** en **writeString** werken met tekst tot aan, niet tot en met, de CR/LF-tekens.

## TextStreams en Strings

TextStreams en Strings zijn verschillende objecten, maar zijn wel verbonden. TextStream-methodes handelen het transport van tekst (invoer/uitvoer) tussen uw applicatie en een schijfbestand af. String-methodes manipuleren tekst, zoals in het volgende voorbeeld wordt getoond.

```
method pushButton(var eventInfo Event)
   var
      tekst1 TextStream
      reeks1, reeks2, reeks3 String
   endVar

   reeks1 = "In barre tijden "
   ; achttien tekens, inclusief cr/lf en één spatie na tijden
   reeks2 = "moet het roer echt om"
   ; 23 tekens, inclusief cr/lf aan einde van regel

   tekst1.create("dries.txt")  ; maakt bestand dries.txt in werkdir
   tekst1.writeLine(reeks1)    ; laat bestand open voor schrijven
   tekst1.writeLine(reeks2)
   tekst1.close()

   if not tekst1.open("dries.txt", "W") then ; opent bestand voor lezen en
                                             ; schrijven
      errorShow()
   endIf
   tekst1.setPosition(28)
   ; volgende lees- en schrijfhandeling begint bij het 28ste
   ; teken in het bestand, de "r" in roer

   reeks3 = "rumoer echt om"
   tekst1.writeString(reeks3)
   tekst1.close()

endMethod
```

DEEL IV

# Speciale onderwerpen

In dit deel van de handleiding worden speciale onderwerpen voor ervaren programmeurs behandeld.

- In Hoofdstuk 18, "Multi-user applicaties", wordt beschreven hoe u applicaties maakt voor gebruik in een netwerk.
- In Hoofdstuk 19, "Fouten en foutverwerking", wordt uitgelegd hoe u fouten behandelt.
- In Hoofdstuk 20, "Scripts maken en afspelen", wordt uitgelegd hoe u ObjectPAL-scripts maakt en afspeelt.
- In Hoofdstuk 21, "ObjectPAL-applicaties samenstellen en aanmaken", wordt beschreven hoe u ObjectPAL-applicaties aanmaakt voor eindgebruikers.

HOOFDSTUK 18

# Multi-user applicaties

In dit hoofdstuk wordt beschreven hoe u applicaties ontwerpt voor een multi-user omgeving (of netwerk). Hoewel multi-user applicaties niet veel verschillen van applicaties op een lokaal station, vragen dergelijke applicaties om extra planning en moet u rekening houden met wat andere gebruikers kunnen doen met de tabellen, formulieren en andere bestanden van uw applicatie.

In dit hoofdstuk worden de voornaamste onderwerpen beschreven en wordt het volgende uitgelegd:

- Hoe u multi-user netwerkapplicaties opzet
- Hoe u privé-directories gebruikt
- Hoe u wachtwoorden voor toegangscontrole gebruikt
- Hoe u vergrendelingen voor toegangscontrole gebruikt
- Hoe u de communicatie tussen processen regelt
- Hoe u werkt en programmeert met aspecten van multi-user applicaties

## Multi-user netwerkapplicaties opzetten

Inherent aan alle multi-user applicaties is het evenwicht tussen de behoefte aan integriteit van de gegevens en de noodzaak om veel gebruikers gelijktijdig toegang te verschaffen tot dezelfde gegevens. U kunt de betrouwbaarheid van gegevens garanderen door slechts één gebruiker tegelijk een tabel te laten gebruiken, maar dat is lastig voor andere gebruikers. Een ander uiterste is dat alle gebruikers tegelijkertijd toegang hebben tot een tabel. Hierdoor zouden de gegevens echter al gauw niet meer betrouwbaar zijn. Het doel van alle multi-user applicaties is de integriteit van gegevens te garanderen bij een zo groot mogelijk aantal gebruikers.

Een belangrijk punt bij het ontwerpen van multi-user applicaties is de afhandeling van gelijktijdige verzoeken om dezelfde resources. Een belangrijk principe hierbij is de gelijktijdige verzoeken om dezelfde resources zo veel mogelijk te beperken en zo efficiënt mogelijk te beheren. Hiervoor gelden drie algemene regels:

- Houd kritieke onderdelen kort: maak gezamenlijke resources niet langer dan nodig is.
- Gebruik de zwakst mogelijke vergrendelingen: vergrendel bijvoorbeeld niet een volledige tabel als de vergrendeling van één record voldoende is.
- Plaats vergrendelingen aan het begin van een methode of procedure: voer bijvoorbeeld geen voorbereidende code uit om vervolgens te ontdekken dat u geen toegang hebt tot de tabel.

## De planning van een multi-user applicatie

Als u een multi-user applicatie opzet, bepaalt u eerst waar de resources die de applicatie gebruikt, zich in het netwerk zullen bevinden. Bevinden alle tabellen zich bijvoorbeeld op een centrale server of worden deze over meerdere servers verdeeld?

Daarna bepaalt u of de applicatie meerdere niveaus nodig heeft voor de beperking van toegang tot gegevens. Zijn er verschillende klassen gebruikers met verschillende behoeften en beveiligingsvereisten? Welke toegangsrechten (lezen, bijwerken, geen) voor de gegevens zijn geschikt voor welke groep gebruikers? Structureer de toegang tot gegevens zodanig dat deze op een logische manier afhankelijk wordt gemaakt van de behoefte en pas het systeem van beveiliging met wachtwoorden hieraan aan.

Ten derde moet u weten wat de vergrendelingseisen zijn voor alle tabellen die de applicaties gebruikt. Volg hierbij het grondbeginsel dat hierboven is genoemd.

Vervolgens bekijkt u de aspecten van het systeembeheer. Gebruikers hebben de juiste toegangsrechten van het besturingssysteem nodig voor directories en bestanden die de applicatie gebruikt. Hoe deze rechten worden toegekend, hangt af van het netwerk waarin uw applicatie actief is. Bepaal hoe nieuwe gebruikers van uw applicatie de juiste toegangsrechten krijgen en hoe u de gebruikers toegangsrechten ontneemt als die niet meer nodig zijn.

Gebruikers moeten vooral lees-, schrijf-, maak- en verwijderrechten krijgen voor de directories waarin tabellen zijn opgeslagen. Als Paradox een tabel vergrendelt, wordt er namelijk een speciaal vergrendelingsbestand (met de extensie LCK) gemaakt in dezelfde directory als waarin de tabel zich bevindt.

Om deze problemen op te lossen heeft de programmeur de volgende belangrijke hulpmiddelen voor gezamenlijk gebruik van informatie tot zijn beschikking:

- Privé-directories, waardoor conflicten tussen tijdelijke tabellen en dergelijke worden voorkomen
- Beveiliging door wachtwoorden, waardoor strikte toegangscontrole mogelijk is
- Vergrendelingen, waardoor de integriteit van gegevens gemakkelijker is te waarborgen
- Communicatie tussen processen die voor sommige applicaties is vereist

In de volgende paragrafen worden deze hulpmiddelen beschreven.

## Privé-directories

Naast het gezamenlijk gebruik van objecten, zoals bestanden en directories, hebben de gebruikers van uw applicatie meestal een of meer privé-objecten (vooral bestanden) nodig. Als een gebruiker Paradox start, krijgt hij automatisch een privé-directory toegewezen. Bovendien kan elke Paradox-sessie een eigen privé-directory opgeven. In beide gevallen verwijst Paradox naar deze directory met de alias :PRIV:.

Paradox slaat de tabel *Antwrd* en andere tijdelijke tabellen die voor een afzonderlijke gebruiker worden gemaakt op in de privé-directory van die gebruiker. Hiermee wordt voorkomen dat gebruikers elkaars tijdelijke bestanden overschrijven.

Gok niet naar de naam van deze privé-directory in uw applicatie. U kunt verwachten dat verschillende gebruikers privé-directories op verschillende plaatsen en met verschillende padnamen hebben. Gebruik daarom altijd de alias :PRIV: om toegang te krijgen tot een tabel in de privé-directory van de gebruiker.

Gebruik de privé-directory om tijdelijke tabellen of dummy-tabellen die in uw applicatie worden gebruikt, op te slaan. De volgende code opent bijvoorbeeld een venster van de tabel *Antwrd* in de privé-directory van de gebruiker en maakt een tijdelijke kopie:

```
var
   antTV TableView
   tmpTbl Table
   antTbl Table
endVar
   antTbl.attach(":PRIV:Antwrd.db")
   tmpTbl.attach(":PRIV:TMPAntw.db")
   tmpTbl.empty()
```

```
antTbl.add(tmpTbl)
antTbl.unattach()
tmpTbl.unattach()

antTV.open(":PRIV:Antwrd.db")
```

Zie Hoofdstuk 16 voor meer informatie over het gebruik van aliassen in ObjectPAL-code en het *Handboek* voor meer informatie over het interactieve gebruik van aliassen.

# Toegang beheren met wachtwoorden

Bij multi-user applicaties moeten verschillende gebruikers vaak op verschillende niveaus toegang krijgen tot gegevens. Sommige gebruikers mogen bijvoorbeeld gegevens wel bekijken maar niet wijzigen. Andere gebruikers mogen bestaande records veranderen, maar niet verwijderen en geen nieuwe records invoeren, terwijl weer andere gebruikers alle wijzigingen mogen aanbrengen die ze willen.

Tabellen zijn standaard niet beveiligd. Paradox kent vijf niveaus van beveiliging door wachtwoorden (zie Tabel 18-1).

Tabel 18-1 Wachtwoordniveaus met beschrijvingen

| Niveau | Beschrijving |
|---|---|
| Alleen lezen | De gebruiker kan uit de tabel lezen |
| Wijzigen | De gebruiker kan gegevens invoeren en wijzigen |
| Invoegen | De gebruiker kan nieuwe records toevoegen |
| Invoegen/ verwijderen | De gebruiker kan records toevoegen en verwijderen |
| Alle | De gebruiker kan alle bewerkingen uitvoeren |

Beveiliging door wachtwoorden wordt meestal interactief beheerd met Paradox. Zie het *Handboek* voor meer informatie. ObjectPAL verschaft bovendien de beheerfuncties die hier worden beschreven.

## Tabel beveiligen

Voordat toegangsbeheer op basis van wachtwoorden mogelijk is, moet de tabel op een van de volgende manieren worden beveiligd:

- Interactief, als de tabel wordt gemaakt of geherstructureerd
- Vanuit ObjectPAL, als de tabel wordt gemaakt
- Vanuit ObjectPAL, door **protect** te gebruiken op een bestaande tabel

De **protect**-methode van het Table-type codeert de tabel en wijst er een gebruikerswachtwoord aan toe. De eigenaar krijgt volledige toegangsrechten ("Alle"), met inbegrip van het recht om de tabel te

herstructureren. Doordat **protect** de tabel codeert met een wachtwoord, kunnen gebruikers geen toegang tot de tabel krijgen met een externe debugger of een tekstbewerker. De **unprotect**-methode van het Table-type maakt deze bewerking weer ongedaan.

### *Toegangsniveaus toewijzen*

Als de tabel is beveiligd, kunnen op de volgende manieren verschillende niveaus van beveiliging door wachtwoorden worden toegewezen:

- Interactief, als de tabel wordt gemaakt of geherstructureerd
- Vanuit ObjectPAL, als de tabel wordt gemaakt

ObjectPAL kent geen manier om de wachtwoordbeperking voor een bestaande tabel te veranderen. Dit is namelijk een interactieve beheerfunctie van het systeem.

### *Toegangsniveaus bepalen*

Als de beveiliging met wachtwoorden eenmaal tot stand is gebracht, moet uw applicatie de toegangsrechten misschien tijdens de uitvoering kunnen vaststellen. U gebruikt hiervoor de methodes **tableRights** en **familyRights** van het Table-type. Deze methodes geven toegangsinformatie over een tabel terug.

### *Toegang krijgen tot beveiligde objecten*

Als een tabel is beveiligd en er wachtwoorden aan zijn toegewezen, wordt de toegang tot de tabel in de ObjectPAL-omgeving verkregen met de **addPassword**-methode van het Session-type. **addPassword**, zo genoemd om redenen die hun oorsprong hebben in het verleden, markeert de tabel als toegankelijk voor deze sessie, op het niveau dat het eerder ingestelde wachtwoord toestaat.

In de meeste applicaties krijgt de gebruiker toegang tot beveiligde tabellen door een wachtwoord op applicatieniveau op te geven, waarna achter de schermen **addPassword** wordt aangeroepen om de gebruiker toegang te geven tot de beveiligde tabel.

De **removePassword**-methode van het Session-type maakt deze handeling ongedaan door de toegang tot de beveiligde tabel voor deze sessie te annuleren.

## Hoe werken sessies?

De toegangscontrole met behulp van wachtwoorden werkt op basis van afzonderlijke sessies, waardoor de applicatieprogrammeur veel mogelijkheden heeft. Zoals u weet, maakt Paradox automatisch een sessie als de applicatie wordt aangeroepen en verschaft ObjectPAL mogelijkheden om aanvullende sessies te maken tijdens de uitvoering.

Stel dat de volgende code is gekoppeld aan de **pushButton**-methode van een knop op een leeg formulier. Deze code beveiligt de tabel *Klant* en wijst het wachtwoord "data" toe. Vervolgens probeert de code tweemaal een TCursor te openen, zonder eerst het wachtwoord

te geven. Eerst gebeurt dit op een Table-variabele en vervolgens op de tabelnaam. Beide pogingen mislukken, omdat de tabel nu met een wachtwoord is beveiligd.

Een aanroep van **addPassword** geeft vervolgens het goede wachtwoord om toegang te krijgen tot de tabel. Dat wil zeggen, "data" wordt toegevoegd aan de lijst met wachtwoorden die deze sessie kent. De tabel kan nu wel worden geopend. De aanroep van **removePassword** verwijdert "hoi" uit de wachtwoordenlijst die deze sessie kent. Als u een volgende keer op deze knop drukt, krijgt u hetzelfde resultaat: eerst gaat het mis en daarna lukt het.

```
method pushButton(var eventInfo Event)
   var
      klantTbl Table
      klantTC TCursor
      wwd String
   endVar

   klantTbl.attach("klant.db")
   klantTbl.protect("hoi")           ; beveilig de tabel

   ; De eerste poging om de tabel te openen
   message(klantTC.open(klantTbl))   ; geeft False weer
    sleep(1111)
   message("")

   ; voer het wachtwoord in
   addPassword("hoi")                ; voer het wachtwoord in

   ; de tweede poging om de tabel te openen
   message(klantTC.open(klantTbl))   ; geeft True weer
   sleep(1111)
   message("")

   removePassword("hoi")             ; verwijdert wachtwoord
endmethod
```

# Vergrendelingen beheren

De automatische vergrendelingsfuncties van Paradox vormen een evenwicht tussen het streven naar integriteit van de gegevens en toegang van meerdere gebruikers bij interactief gebruik. Deze functies helpen u hetzelfde evenwicht te bereiken in uw applicaties. Bovendien kunt u met behulp van deze functies een functionele multi-user applicatie schrijven, zonder ooit expliciet records of objecten te hoeven vergrendelen.

Gezamenlijk gebruikte resources moet u normaal gesproken echter wel expliciet vergrendelen, zodat uw applicatie de juiste handelingen kan uitvoeren als een vergrendeling mislukt. Tegen deze achtergrond wordt in deze paragraaf uitgelegd hoe vergrendelingen werken in Paradox.

*Hoe werken vergrendelingen?*

Vergrendelingen beperken de toegang tot een tabel of een record. Vergrendelingen kunnen automatisch door Paradox worden geplaatst als een bepaalde bewerking wordt aangeroepen, of expliciet worden geplaatst in een ObjectPAL-methode.

Vergrendelingen en wachtwoordcontrole verschillen op de volgende punten van elkaar: wachtwoorden beperken de toegang vanaf het begin, nog voordat er tabellen zijn geopend. Wachtwoorden beperken de toegang op basis van toegangsrechten van gebruikers. Vergrendelingen werken op een rechtstreekse, real-time basis en pas nadat via een wachtwoord toegang is verstrekt tot resources. Vergrendelingen geven de programmeur tijdelijk de controle over gegevens-resources. Dit gebeurt naar behoefte en gebruik en niet op basis van een wachtwoord. U kunt vergrendelingen het best gebruiken om alleen kleine codesegmenten die toegang geven tot of wijzigingen aanbrengen in gegevens, af te schermen, terwijl u wachtwoorden gebruikt om grote delen van een applicatie of de hele applicatie af te schermen.

In deze paragraaf worden de volgende onderwerpen behandeld:

- Automatische vergrendelingen
- Expliciete vergrendelingen
- Types tabelvergrendelingen
- Werken met 'lock' en 'unlock'
- Records vergrendelen

## Automatische vergrendelingen

In een multi-user omgeving vergrendelt Paradox objecten automatisch tijdens bewerkingen waarbij conflicten kunnen optreden omtrent een resource. In het algemeen zijn de vergrendelingen die Paradox automatisch plaatst, zo zwak als mogelijk is zonder dat de integriteit van de gegevens tijdens de bewerking in gevaar komt.

Als u bijvoorbeeld **copy** gebruikt om een tabel te kopiëren, plaatst Paradox een schrijfvergrendeling op de brontabel. De schrijfvergrendeling voorkomt dat andere gebruikers de brontabel tijdens het kopiëren veranderen, maar voorkomt geen alleen-lezen handelingen, zoals weergeven. Paradox plaatst ook een volledige vergrendeling op de doeltabel die bij de kopieerprocedure is betrokken. Deze vergrendeling voorkomt dat andere gebruikers tijdens de kopieerprocedure op de een of andere manier toegang krijgen tot de doeltabel.

*Niet-bestaande resources vergrendelen*

In dit geval bestaat de doeltabel vaak niet aan het begin van de bewerking. Toch kan Paradox deze doeltabel vergrendelen. De mogelijkheid om een resource die niet bestaat, te vergrendelen is essentieel, omdat zo wordt voorkomen dat een andere gebruiker een

object maakt tijdens de kopieerprocedure of een tabel verwijdert die u net hebt gemaakt, maar nog niet eens hebt gebruikt.

Automatische vergrendelingen worden ook gebruikt bij de meeste batch-achtige bewerkingen, zoals **add**, **subtract**, **cSum**, **cNpv** enzovoort.

## Expliciete vergrendelingen

Hoewel Paradox voor veel bewerkingen automatische vergrendelingen gebruikt, kunt u in de meeste multi-user applicaties beter expliciete vergrendelingsopdrachten gebruiken om de toegang tot resources te beheren. Dit is beter dan afhankelijk te zijn van automatische vergrendelingen. De expliciete vergrendelingsopdrachten geven u meer controle dan de automatische vergrendelingen en maken het ook gemakkelijker te reageren op situaties waarin een gebruiker geen vergrendeling kan plaatsen vanwege een resource-conflict.

Expliciete en automatische vergrendelingen kunnen tegelijkertijd actief zijn in dezelfde tabel. Als u bijvoorbeeld de **lock**-methode van het Session-type gebruikt om expliciet een volledige vergrendeling te plaatsen op de tabel *Order* en vervolgens de **copy**-methode van het Table-type gebruikt om *Order* te kopiëren naar *Nweords*, hebt u zowel een expliciete volledige vergrendeling als een automatische schrijfvergrendeling op *Order* geplaatst. Voor de andere gebruikers lijkt het alsof op *Order* alleen een volledige vergrendeling is geplaatst, omdat dit de sterkste van de twee vergrendelingen is. Aan het eind van de bewerking verdwijnt de automatische schrijfvergrendeling, terwijl uw expliciete volledige vergrendeling intact blijft.

Andere gebruikers kunnen ook zowel expliciete als automatische vergrendelingen plaatsen op objecten die u hebt vergrendeld, zolang deze vergrendelingen legaal kunnen bestaan naast de bestaande vergrendelingen. Pogingen om automatische en expliciete vergrendelingen te plaatsen op objecten en records worden afgehandeld volgens het principe: 'wie het eerst komt, die het eerst maalt'.

## Types tabelvergrendelingen

De types tabelvergrendelingen die beschikbaar zijn in Paradox, variëren in sterkte en in de mate waarin gezamenlijk gebruik is toegestaan. Deze types worden gedetailleerder besproken in latere paragrafen. Table 12-2 toont de vergrendelingen die u kunt plaatsen, in volgorde van *afnemende* sterkte en *toenemende* mogelijkheden voor gezamenlijk gebruik.

Tabel 18-2 Vergrendelingstypes

| Vergrendelingstype | Beschrijving |
|---|---|
| Volledige vergrendeling | Geen andere sessie kan de vergrendelde tabel lezen, schrijven of herstructureren |
| Schrijfvergrendeling | Geen andere sessie kan een schrijf- of leesvergrendeling op deze tabel plaatsen |
| Leesvergrendeling | Geen andere sessie kan een schrijfvergrendeling of een volledige vergrendeling op deze tabel plaatsen |

U kunt *volledige* vergrendelingen plaatsen op Table-variabelen, TCursor-variabelen of op tabellen waarnaar u verwijst met namen tussen aanhalingstekens. *Schrijfvergrendelingen* en *leesvergrendelingen* kunt u echter alleen op TCursor-variabelen plaatsen. Daarnaast bestaat er een aantal specifieke verschillen tussen Paradox- en dBASE-tabellen, zoals in Tabel 18-3 wordt getoond.

Tabel 18-3 Beschikbare vergrendelingen voor Paradox- en dBASE-tabellen en TCursors

| | TCursor-var | Table-var | Tabelnaam tussen aanhalingstekens |
|---|---|---|---|
| Lezen | P | P* | P* |
| Schrijven | P D | P* D* | P* D* |
| Volledig | P D | P D* | P D* |

P = Paradox-tabel D = dBASE-tabel *als tabel bestaat

Volledige vergrendelingen en schrijfvergrendelingen beperken de bewerkingen die andere gebruikers op de tabel kunnen uitvoeren. Deze vergrendelingen zijn geschikt als een tabel gedurende een bepaalde periode stabiel moet blijven. Zo is het verstandig een schrijfvergrendeling op een tabel te plaatsen als u een bewerking wilt uitvoeren die niet toestaat dat andere gebruikers wijzigingen in de tabel aanbrengen, zoals het wijzigen van de inhoud van bepaalde records.

U gebruikt een volledige vergrendeling als u het exclusieve gebruik van een tabel wilt. U gebruikt bijvoorbeeld een volledige vergrendeling als een applicatie de volgende bewerkingen moet uitvoeren:

- Werken met vastliggende informatie in een tijdelijke tabel
- Een tabelnaam reserveren terwijl de tabel nog niet is gemaakt
- Een of meer wijzigingen aanbrengen in de structuur van een tabel, zoals maken, verwijderen, sorteren of herstructureren

- Verscheidene informatiemethodes op de tabel gebruiken, zoals **cMax** en **cNpv**, waarbij het zeker moet zijn dat de inhoud niet verandert

Veel van deze bewerkingen plaatsen deze vergrendelingen overigens automatisch.

## lock en unlock

U gebruikt de methodes **lock** en **unlock**, die beide zowel voor het Table-type als voor het TCursor-type zijn gedefinieerd, om expliciet vergrendelingen te plaatsen op een of meer tabellen. Zowel **lock** als **unlock** nemen als argumenten een lijst van een of meer door komma's gescheiden reeksen die tabelnamen en types vergrendelingen vertegenwoordigen. De reeksen FULL, READ en WRITE worden gebruikt voor respectievelijk volledige vergrendelingen, leesvergrendelingen en schrijfvergrendelingen.

De volgende instructies plaatsen een volledige vergrendeling op de tabel *Order* en een schrijfvergrendeling op de tabel *Voorraad*:

```
var
   orderTbl Table
   voorraadTC TCursor
endVar

orderTbl.attach("order.db")
voorraadTC.open("voorraad.db")
lock(orderTbl,"FULL",voorraadTC,"WRITE")
```

Als de **lock**-methode in één instructie vraagt om vergrendelingen op een lijst met tabellen, worden alle vergrendelingen geplaatst of geen enkele. Deze eigenschap voorkomt bepaalde soorten impasses bij de toegang tot tabellen. **lock** geeft True terug als alle vergrendelingen met succes kunnen worden geplaatst en False als dat niet het geval is. Gebruik in het laatste geval de System-procedures voor de foutstapel, **errorCode**, **errorMessage** en **errorShow** om vast te stellen waarom de bewerking is mislukt.

Een goed voorbeeld van een applicatie waarin records in verschillende tabellen tegelijkertijd moeten worden vergrendeld, is een systeem voor orderverwerking. Voor een bepaalde klant moet u misschien als onderdeel van dezelfde transactie inventaris- en factuurtabellen bijwerken. Als u deze tabellen ten opzichte van elkaar kloppend wilt houden, moet u in beide tabellen het juiste record vergrendelen voordat u een van de records wijzigt.

### Bereik van vergrendelingen

Vergrendelingen worden *niet* vrijgegeven als de methode die de vergrendelingen heeft plaatst, is afgelopen. Volledige vergrendelingen blijven bestaan totdat deze expliciet worden vrijgegeven of totdat de sessie eindigt. Hoe lang schrijf- en leesvergrendelingen blijven bestaan, is afhankelijk van het bereik van de vergrendelde variabele.

Leesvergrendelingen garanderen toegang tot een consistente verzameling gegevens. Hoewel andere sessies de gegevens kunnen bekijken, voorkomt deze vergrendeling dat andere sessies schrijfvergrendelingen en volledige vergrendelingen op de tabel plaatsen. Als u een leesvergrendeling plaatst op de tabel die u wilt bewerken, voorkomt u dat andere gebruikers de gegevens veranderen terwijl u deze gebruikt. Een leesvergrendeling voorkomt niet dat andere gebruikers een record vergrendelen dat u wilt bewerken.

### Volledige vergrendelingen plaatsen

Een volledige vergrendeling voorkomt dat andere gebruikers op welke manier dan ook toegang krijgen tot de tabel. U hebt als enige toegang. Als u een volledige vergrendeling op een tabel plaatst, kan een andere gebruiker deze tabel niet kiezen uit een Paradox-menu of toegang tot de tabel krijgen via een ObjectPAL-methode. Als iemand anders de tabel heeft geopend of vergrendeld, wordt deze vergrendeling geweigerd. Een volledige vergrendeling wil dus zeggen dat geen enkele andere gebruiker in een andere sessie uit deze tabel kan lezen of naar deze tabel kan schrijven.

***Opmerking*** Wanneer u een volledige vergrendeling plaatst op een tabel is dit anders dan wanneer u de methode **setExclusive** aanroept die is gedefinieerd voor het Table-type. Bepaalde bewerkingen, zoals herstructureren, kunnen alleen worden uitgevoerd als u **setExclusive** aanroept. Een volledige vergrendeling geeft niet voldoende rechten.

De volledige vergrendeling komt overeen met de volledige vergrendeling uit oudere versies van Paradox. U kunt geen volledige vergrendeling op een dBASE-tabel plaatsen.

Een volledige vergrendeling kan worden geplaatst op een Table-variabele, een TCursor-variabele of een tabelnaam tussen aanhalingstekens. Deze vergrendeling blijft bestaan totdat u deze opheft of de sessie beëindigt waarin de vergrendeling werd geplaatst. In het volgende voorbeeld wordt **lock** gebruikt met een Table-variabele en een tabelnaam om volledige vergrendelingen te plaatsen op de tabel *Verkoop* en de tabel *Order*.

```
var verkoopTbl Table endVar
   verkoopTbl.attach("verkoop.db")
   IF NOT lock(verkoopTbl, "FULL", "order.db", "FULL") then
      ; strategie voor afhandeling van resources...
   ENDIF
```

### Schrijfvergrendelingen plaatsen

Een schrijfvergrendeling voorkomt dat andere gebruikers de inhoud van een tabel veranderen of een schrijfvergrendeling of een volledige vergrendeling op een tabel plaatsen. Een schrijfvergrendeling beperkt niet de mogelijkheid om de tabel te bekijken en beperkt ook niet de toegang tot objecten in de verwanten van de tabel. Als een andere gebruiker een volledige vergrendeling, een schrijfvergrendeling of een

leesvergrendeling op de tabel heeft geplaatst, wordt deze schrijfvergrendeling geweigerd.

Een schrijfvergrendeling zegt dus: "Ik schrijf naar deze tabel; geen andere gebruiker mag dit doen; ik heb heb als enige schrijftoegang nodig om de integriteit van de gegevens te behouden".

### Leesvergrendelingen plaatsen

Een leesvergrendeling voorkomt dat andere gebruikers expliciet of automatisch een schrijfvergrendeling of een volledige vergrendeling op een tabel plaatsen, maar voorkomt niet dat andere sessies ook leesvergrendelingen plaatsen op de tabel. Als er al een schrijfvergrendeling op de tabel is geplaatst, wordt een verzoek om een leesvergrendeling geweigerd. Een leesvergrendeling voorkomt niet dat andere sessies leesvergrendelingen plaatsen. Paradox waardeert leesvergrendelingen op dBASE-tabellen, op tot schrijfvergrendelingen. In Paradox komt deze leesvergrendeling overeen met het beletten van een schrijfvergrendeling in oudere versies van Paradox.

Een leesvergrendeling zegt dus : "Ik werk in deze tabel; andere gebruikers kunnen geen schrijfvergrendeling plaatsen op deze tabel of de gegevens veranderen".

U gebruikt een TCursor om een leesvergrendeling te plaatsen. De code in het volgende voorbeeld plaatst een leesvergrendeling op de tabel *Order*.

```
var orderTC TCursor endVar
lock(orderTC, "READ")
```

### Impasses vermijden

In bepaalde situaties kan de vraag om resources in een multi-user omgeving tot een impasse leiden. Stel dat twee gebruikers, Ben en Dick, allebei de tabellen *Order* en *Voorraad* willen vergrendelen. Ben heeft *Order* met succes vergrendeld en probeert nu *Voorraad* te vergrendelen. Dick heeft ondertussen *Voorraad* vergrendeld en probeert nu *Order* te vergrendelen. Beide gebruikers wachten op een resource die door de andere wordt vastgehouden. Hierdoor ontstaat een *impasse*.

Omdat u met **lock** meer dan één tabel tegelijk kunt vergrendelen, kent deze methode ingebouwde manieren om een dergelijk impasse te voorkomen. Stel dat Ben de volgende code uitvoert:

```
var
   Order, Voorraad Table
endVar
Order.attach("order.db")
Voorraad.attach("voorraad.db")
lock(Order, "FULL", Voorraad, "FULL")
```

Stel dat Dick tegelijkertijd deze code uitvoert:

```
var
   Voorraad, Order Table
```

```
endVar
Voorraad.attach("voorraad.db")
Order.attach("order.db")
lock(Voorraad, "FULL", Order, "FULL")
```

Een van de twee vergrendelt beide tabellen terwijl de ander moet wachten.

U kunt een impasse vermijden door alle resources die u voor een bewerking nodig hebt, met één **lock**-instructie te vergrendelen. Als u uw applicaties structureert door kleine modules te gebruiken en alle tabellen die u nodig hebt in één keer vergrendelt, hoeft u zich geen zorgen te maken over een mogelijke impasse.

### *Meerdere vergrendelingen op een tabel plaatsen*

U kunt expliciet twee of meer verschillende types vergrendelingen tegelijkertijd op dezelfde tabel plaatsen. De code in het volgende voorbeeld probeert een schrijfvergrendeling en een leesvergrendeling op de tabel *Order* te plaatsen:

```
var orderTC TCursor endVar
orderTC.open("order.db")
lock(OrderTC, "WRITE", OrderTC, "READ")
```

Als u meerdere vergrendelingen op één tabel toepast, moet u elke expliciete vergrendeling koppelen aan een expliciete ontgrendeling van hetzelfde type. Vergrendelingen kunnen worden gestapeld en een object wordt pas ontgrendeld als u alle vergrendelingen die u hebt geplaatst, weer hebt verwijderd.

Als u twee of meer vergrendelingen van hetzelfde type op één tabel plaatst, verandert het effect van de vergrendeling niet. Meerdere vergrendelingen van hetzelfde type kunnen soms echter het besturingsverloop in programma's vereenvoudigen. U kunt bijvoorbeeld methodes of procedures nesten die elk een **lock**, uitvoeren gevolgd door een **unlock**, zonder dat de vergrendelingen die al zijn geplaatst door de aanroepende methode of procedure, ongedaan worden gemaakt.

## Testen op geslaagde vergrendeling

Test altijd op de waarde die **lock** teruggeeft of gebruik **errorCode** nadat u hebt geprobeerd expliciete vergrendelingen te plaatsen, tenzij u zeker weet dat de poging niet mislukt. Het is vaak gemakkelijk om de vergrendeling en de bijbehorende test of **errorCode** in een **while**-lus te plaatsen, zoals in het volgende voorbeeld:

```
var
   emp TCursor
endvar
emp.open("werknmr.db")
while True
   if lock(emp, "FULL") then  ; vergrendeling geslaagd?
      quitLoop                ; ga dan verder achter lus
   else
      msgInfo(errorCode(), errorMessage()) ; toon foutmelding aan gebruiker
  endIf
```

```
  ...                              ; rest van methode
endWhile
```

Wat u in de **else**-clausule van deze code plaatst, hangt af van de applicatie. Het kan bijvoorbeeld netjes zijn gebruikers te vragen om te wachten totdat de tabel vrij is. Als u de applicatie zo hebt gestructureerd dat het oponthoud kort duurt, kunt u alleen een bericht **Wachten...** weergeven, een seconde wachten met de System-procedure **sleep** en vervolgens teruggaan naar het begin van de **while**-clausule. Ga niet terug naar het begin van de lus zonder eerst te pauzeren, want dan wordt het netwerk onnodig belast. U kunt ook **setRetryPeriod** (Session-type, zie verderop in dit hoofdstuk) gebruiken om een automatische herhalingsperiode in te bouwen in de poging een vergrendeling te plaatsen in de sessie.

Als een expliciete vergrendeling niet langer nodig is, gebruikt u **unlock** om deze vrij te geven. Stel dat de tabellen *Order* en *Voorraad* zijn vergrendeld met de volgende instructies:

```
var
   voorraadTC TCursor
   orderTbl Table
endVar
voorraadTC.open("voorraad.db")
orderTbl.attach("order.db")
lock(orderTbl, "FULL", voorraadTC, "WRITE")
```

Als u de vergrendelingen ongedaan wilt maken, roept u **unlock** aan en geeft u de tabellen en de types vergrendelingen op die u wilt ontgrendelen.

```
var
   voorraadTC TCursor
   orderTbl Table
endVar
unlock(orderTbl, "FULL", voorraadTC, "WRITE")
```

### *lockStatus*

U gebruikt **lockStatus** om te bepalen of u een bepaald type vergrendeling expliciet op een tabel hebt geplaatst, en zo ja, hoe vaak. De volgende instructie geeft bijvoorbeeld het aantal schrijf-vergrendelingen terug dat is geplaatst op de tabel *Order*:

```
lockStatus("Order.db", "WRITE")
```

Met de reeks ANY als tweede argument kunt u bepalen hoeveel vergrendelingen van een willekeurig type u op de tabel hebt geplaatst.

**lockStatus** geeft alleen uw eigen vergrendelingen terug en niet die van andere gebruikers.

## Records vergrendelen

Wanneer u een recordvergrendeling plaatst, voorkomt u op record-niveau vergrendelt in plaats van op tabelniveau dat andere gebruikers de vergrendelde gegevens van records wijzigen of verwijderen tijdens de duur van de vergrendeling. Vergrendelingen

van records lijken op schrijfvergrendelingen op een tabel, omdat andere sessies het vergrendelde record wel kunnen bekijken, maar niet veranderen en er ook geen tweede recordvergrendeling op kunnen plaatsen. Als een andere gebruiker naar een record gaat dat u hebt vergrendeld, heeft dat record de waarde die het had toen de vergrendeling werd geplaatst. Pas nadat het record is ontgrendeld, worden uw wijzigingen in de tabel doorgevoerd en zijn deze beschikbaar voor andere gebruikers.

Vanuit de applicatie bekeken, dient de vergrendeling van records twee doelen:

- U krijgt de exclusieve mogelijkheid om een bepaald record te veranderen zonder de hele tabel te vergrendelen. Dit kan de prestaties van uw applicatie verbeteren.
- Als een record wordt bijgewerkt, frist Paradox waarden op, zodat deze wijzigingen in andere sessies in het netwerk worden weergegeven.

*lockRecord en unlockRecord*

Net als vergrendelingen van tabellen kunnen vergrendelingen van records automatisch worden geplaatst door een handeling die een record verandert, of expliciet door middel van **lockRecord**. Zo kunnen vergrendelingen van records ook automatisch worden vrijgegeven doordat de cursor naar een ander record wordt verplaatst of expliciet door middel van **unlockRecord**. Expliciete vergrendelingen en ontgrendelingen verdienen de voorkeur vanwege de betere beheersingsmogelijkheden en prestaties.

Als u wilt begrijpen hoe **lockRecord** en **unlockRecord** werken, moet u weten dat er twee elementaire manieren zijn om een record te bewerken:

- U kunt een record veranderen dat al bestaat.
- U kunt een nieuw record invoeren dat nog niet in de tabel is doorgevoerd.

### Bestaande records vergrendelen

Als u in een multi-user omgeving een bestaand record wilt bekijken of wijzigen dat onder programmabesturing staat, maakt u van het record het huidige record en voert u vervolgens de **lockRecord**-methode uit vanuit een TCursor of een UIObject .

Meestal kunt u het beste testen op de teruggegeven waarde van **lockRecord** om te zien of de vergrendeling is geplaatst. Als de vergrendeling mislukt, gebruikt u **errorCode** en **errorMessage** om te bepalen hoe dat komt. De meest waarschijnlijke reden is dat er al een andere vergrendeling op het record is geplaatst. In dat geval geven de foutstapelberichten aan welke sessie de vergrendeling heeft geplaatst. De vergrendeling kan ook zijn mislukt omdat een andere gebruiker

een schrijfvergrendeling heeft geplaatst op de tabel die het record bevat of omdat het record is verwijderd en dus niet meer bestaat.

Als de vergrendeling slaagt, kan de applicatie doorgaan met het bekijken of wijzigen van het record. Als de verwerking van het record is afgerond, gebruikt u **unlockRecord** om het record te ontgrendelen en eventuele wijzigingen door te voeren in het record.

Net als **lockRecord** geeft **unlockRecord** True terug bij succes en False bij een mislukking. Een mislukking kan worden veroorzaakt door een inbreuk op een sleutelveld.

### Nieuwe records invoeren en doorvoeren

Als u een nieuw, leeg record maakt en gegevens invoert in dit record, bestaat dit record met de gegevens pas in de tabel als het is doorgevoerd. Houd er ook rekening mee dat een nieuw record dat nog niet is doorgevoerd, niet kan worden vergrendeld. Als een nieuw record nog niet is doorgevoerd, hoeft het niet te worden vergrendeld, omdat andere gebruikers er geen toegang toe kunnen krijgen. Als het record in de tabel kan worden doorgevoerd, gebruikt u **postRecord** of verplaatst u de cursor uit het record.

## Communicatie tussen processen

In bepaalde multi-user applicaties moet de programmeur de activiteiten tussen processen of applicaties in het netwerk coördineren. In een client-server architectuur kunnen bijvoorbeeld meerdere *client*-verwerkingen tegelijkertijd bijwerkingen doorvoeren in een centrale hoofdtabel. De client die het eerst is geactiveerd, initialiseert normaal gesproken de hele omgeving voor zichzelf en voor alle toekomstige clients. Elke client die zich aanmeldt, moet bepalen of de initialisatieprocedure al is uitgevoerd of dat een ander client-proces op dat moment de initialisatieprocedure uitvoert.

U kunt dit soort problemen oplossen door gebruik te maken van het vermogen van Paradox om de vergrendelingsintegriteit van tabellen te waarborgen. U kunt bijvoorbeeld als volgt te werk gaan:

- Gebruik de inhoud van een gezamenlijk gebruikte tabel om informatie door te geven tussen processen.
- Gebruik de vergrendelingen die op een gezamenlijk gebruikte tabel zijn geplaatst, als vlaggen of seinen voor activiteiten over meerdere processen.

Als u een dergelijke strategie volgt, maakt u een speciale tabel die intern functioneert als informatiesluis wanneer u applicatie meerdere keren is gestart. Informatie kan op elke geschikte manier tussen de processen worden uitgewisseld. Het ene type proces kan bijvoorbeeld

informatie naar een tabel schrijven, terwijl een of meer andere procestypes de lezers kunnen zijn die vergrendelingen op deze tabel gebruiken en misschien een vlaggensysteem voor de afstemming.

Een ander voorbeeld is dat uw applicatie kan bepalen hoeveel client-processen er op het moment actief zijn als elke client bij het opstarten een record naar een centrale tabel schrijft dat aangeeft dat de client bestaat, en dit record weer verwijdert als de client klaar is. Andere processen kunnen op elk moment het totale aantal actieve clients bepalen door **cCount** te gebruiken om het aantal records in de tabel op te vragen.

Hoewel geen enkel ObjectPAL-mechanisme een expliciete uitwisseling van parameters tussen processen ondersteunt, hebben deze technieken hetzelfde resultaat.

# Database-programmering en multi-user applicaties

In dit hoofdstuk wordt een aantal aspecten van database-programmering beschreven die vooral belangrijk zijn voor de ontwikkeling van multi-user applicaties. Als u deze concepten begrijpt, kunt u het gedrag van een applicatie sturen en de gebruikers van uw applicatie precies geven wat ze nodig hebben.

In deze paragraaf worden de volgende onderwerpen behandeld:

- De behandeling van sleutelconflicten
- Wegvliegen van records en relatieve volgorde
- De invloed van **postRecord** en **unlockRecord** op vergrendelingen
- De invloed van tabelvergrendelingen op het gegevensmodel
- Strategieën die Paradox volgt als een record wordt ontgrendeld, doorgevoerd of geannuleerd
- Automatische herhalingen uitvoeren met **setRetryPeriod**
- Het verschil tussen **setExclusive** en **setReadOnly**

***Opmerking*** Deze concepten zijn niet alleen belangrijk bij de ontwikkeling van multi-user applicaties, maar ook handig bij de ontwikkeling van single-user applicaties.

## Sleutelconflicten

Een sleutelconflict ontstaat in de volgende gevallen:

- U probeert een nieuw record door te voeren met dezelfde sleutel als een bestaand record.
- U probeert een wijziging door te voeren in een bestaand record waardoor dat record dezelfde sleutel krijgt als een bestaand record.

In beide gevallen mislukt de doorvoering. Paradox behoudt de vergrendeling op het bestaande record met de sleutel die het conflict heeft veroorzaakt. U kunt de foutstapel bekijken om de oorzaak van de mislukking te bepalen. Er zijn twee hulpmiddelen beschikbaar voor het oplossen van inbreuken op sleutels:

- De TCursor-methode **updateRecord** wijzigt het bestaande record na een inbreuk op een sleutel door dit record bij te werken met waarden van het nieuwe record. Het oude record wordt dus vervangen door het nieuwe. Het maakt geen verschil of het record dat moet worden doorgevoerd, een nieuw record of een bestaand record was. In beide gevallen wordt het record verwijderd. Deze methode geldt alleen voor Paradox-tabellen. De methode gebruikt een optioneel logisch argument om aan te geven of het record met **moveTo** moet worden gevolgd als het wegvliegt.
- De TCursor-methode **attachToKeyViol** gebruikt als argument de TCursor die de inbreuk op de sleutel rapporteerde. Dit is een snelle manier om toegang te krijgen tot het bestaande record met de beoogde sleutel door een tweede TCursor naar het record te laten verwijzen.

## Wegvliegen en relatieve volgorde

Als u ooit records hebt toegevoegd aan tabellen met sleutels, zal u zijn opgevallen hoe nieuwe records naar de juiste posities "sprongen" toen u deze doorvoerde. Deze verplaatsing, die *wegvliegen* wordt genoemd, kan verwarrend zijn als u voor het eerst interactief werkt met tabellen met sleutels. Wegvliegen kan subtiele veranderingen in het gedrag van applicaties veroorzaken, vooral in multi-user omgevingen (waar andere gebruikers records en sleutelwaarden kunnen toevoegen of wijzigen terwijl uw applicatie met een bewerking bezig is).

Wegvliegen is normaal en doet zich voor als de volgorde van de records van een tabel verandert (meestal als een record wordt toegevoegd of als de index wordt gewijzigd). Net als bij andere aspecten van de ontwikkeling van een multi-user applicatie, is in dit geval enige planning nodig voor een goede werking.

Paradox kent het alleen-lezen kenmerk 'FlyAway' voor velden, records, tabelframes, multi-record objecten en formulieren. 'FlyAway' is True als de laatste recordontgrendeling (of recorddoorvoering) er voor zorgde dat het record van plaats veranderde. Als het record op

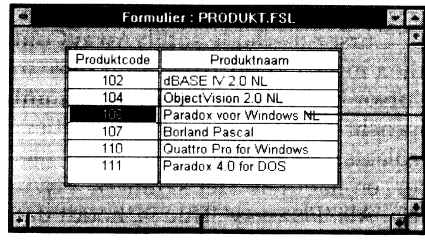
de huidige positie blijft, is 'FlyAway' False. Dit kenmerk is alleen geldig (en nuttig) direct na een **dataUnlockRecord**-handeling of een **dataPostRecord**-handeling, dus direct nadat het kenmerk is ingesteld. De status van het kenmerk blijft ongewijzigd totdat er een nieuwe gelijksoortige handeling wordt uitgevoerd.

Het volgende voorbeeld toont code die is gekoppeld aan de ingebouwde actiemethode van een tabelframe (dit werkt ook bij een multi-record object). De code test op een DataUnlockRecord en forceert een DataPostRecord. Hierdoor wordt de ingebouwde code aangeroepen die het record doet wegvliegen. Het tabelframe volgt hierbij het record.

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataUnlockRecord then
      self.action(DataPostRecord)
   endIf
endMethod
```

Wegvliegen kan worden gedemonstreerd aan de hand van een formulier dat een tabelframe bevat met een sleutel op het veld 'Produktnr.', zoals in Afbeelding 18-1. Als u record 3 selecteert en de sleutelwaarde verandert in 106, geeft het kenmerk 'FlyAway' False terug, omdat het record op dezelfde plaats blijft. Als u de sleutelwaarde verandert in 100, verplaatst het record zich en geeft het kenmerk 'FlyAway' True terug.

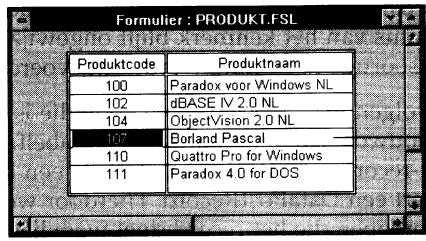
Afbeelding 18-1 Werken met het kenmerk 'FlyAway'




In ObjectPAL kan wegvliegen onduidelijkheid veroorzaken over de vraag waarnaar een TCursor verwijst als een record is doorgevoerd. De algemene regel is dat als een record zich verplaatst, de TCursor naar het volgende record verwijst, zoals in Afbeelding 18-2. Als de sleutelwaarde van het record verandert van 105 in 100, vliegt record 3 weg naar positie 1. De TCursor moet naar een geldig record verwijzen. Daarom wordt (impliciet) de **nextRecord**-methode uitgevoerd en verwijst de TCursor naar record 4.

***Belangrijk*** Als u werkt met tabellen met sleutels en **postRecord** aanroept, volgt de TCursor het doorgevoerde record als het wegvliegt. Als u **unlockRecord** aanroept, volgt de TCursor het record niet.

Afbeelding 18-2 Meer over het kenmerk 'FlyAway'




*Wegvliegen kan voorkomen in scan-lussen*

Let goed op bij wegvliegen in **scan**-lussen waarin de relatieve volgorde van records in de tabel wordt veranderd terwijl u scant. Als u bijvoorbeeld door een tabel scant en bewerkingen soms de relatieve volgorde van records veranderen, wat gebeurt er dan als het actieve record wegvliegt?

```
Proc eenProc(var tc TCursor)
   scan tc:
      if eenVoorwaarde = True then          ; Onder bepaalde voorwaarden,
         tc.delete()                        ; dit record verwijderen.
      endif
      tc.nextRecord()                       ; Naar welk record verwijst tc nu?
   endScan
endProc
```

Als de relatieve volgorde van records verandert, weet u normaal gesproken niet zeker waar de TCursor zich bevindt nadat het record is ontgrendeld. U kunt dit probleem oplossen met de volgende twee methodes:

- De TCursor-methode **setFlyAwayControl** heeft twee effecten: indien ingesteld op True, zorgt deze methode ervoor dat de TCursor bij het record blijft na een **unlockRecord**-methode, ongeacht of het record van de relatieve positie in de tabel wordt verplaatst, zolang het record maar niet verder gaat dan de directe voor- en naburen. Het kenmerk 'FlyAway' wordt ingesteld op False. Als 'Flyaway' True is, houdt dat in dat het record niet van de relatieve positie is verplaatst.
- De TCursor-methode **didFlyAway** geeft na een **unlockRecord** aan of de TCursor inderdaad is weggevlogen.

Deze twee methodes zorgen ervoor dat **scan**-lussen gemakkelijker te begrijpen worden, maar dit gaat ten koste van de prestaties. **setFlyAwayControl** leidt tot veel interne controles na elke cursorbewerking, waardoor de werking wordt vertraagd. Gebruik deze methodes daarom met enige voorzichtigheid.

## postRecord en vergrendelingen

Als u werkt met tabellen met sleutels, voert de TCursor-methode **postRecord** wijzigingen door in het record, zonder het te

ontgrendelen. Het record blijft het huidige record. **unlockRecord** daarentegen ontgrendelt het record en laat het indien nodig wegvliegen naar de nieuwe sorteerpositie in de tabel.

Gebruik bij de ontwikkeling van multi-user applicaties de methode die geschikt is voor de handeling die u wilt laten uitvoeren als records worden doorgevoerd. Hoewel u de **postRecord**-methode het meest zult gebruiken, is **unlockRecord** soms beter.

## Tabelvergrendelingen op het gegevensmodel

Als u een record op een interactieve manier of met ObjectPAL vergrendelt, vergrendelt Paradox automatisch alle verwante tabellen in het gegevensmodel om de referentiële integriteit te behouden. Paradox vergrendelt eerst de bovenste hoofdtabel van het gegevensmodel en gaat vervolgens omlaag naar de tabel die u wilt vergrendelen, waarbij elke tussenliggende hoofdtabel ook wordt vergrendeld. Dit is nodig om te voorkomen dat een andere sessie de hoofdtabel verandert en details daardoor wegvliegen. Dit kan verwarrend overkomen voor niet-ingewijden, omdat het voor een andere sessie lijkt of de hoofdtabel niet kan worden vergrendeld, terwijl de eerste sessie alleen een vergrendeling heeft geplaatst op een detailrecord.

Verwante tabellen die zelf geen hoofdtabellen zijn, worden niet beïnvloed. Dat wil zeggen dat als er twee één-op-meer tabellen zijn voor één hoofdtabel, de vergrendeling van de ene detailtabel geen invloed heeft op de andere.

## Ontgrendelen, doorvoeren en annuleren

Als een record wordt ontgrendeld, doorgevoerd of geannuleerd, volgt Paradox een van de volgende strategieën:

- Als dit gebeurt door middel van een toetsaanslag of een menu-optie (dat wil zeggen interactief), wordt het hele gegevensmodel beïnvloed, zoals is beschreven in "Vergrendelingen beheren", eerder in dit hoofdstuk.
- Als dit een interne actie is of het resultaat van een **action**-methode van ObjectPAL, wordt alleen de tabel beïnvloed die is geassocieerd met het doel van de actie.
- Een ontgrendeling, doorvoering of annulering van een record die direct naar het formulier wordt gestuurd, beïnvloedt het hele gegevensmodel, net als bij de interactieve variant.

Het gevolg hiervan is dat ObjectPAL elke tabel in het gegevensmodel kan ontgrendelen, doorvoeren of annuleren, zonder dat dit invloed heeft op de andere tabellen, terwijl interactief gebruik automatisch het hele gegevensmodel beïnvloedt.

*Opmerking voor dBASE-programmeurs*

Paradox hoeft een record niet expliciet te vergrendelen voordat het wordt verwijderd. (De verwijdering mislukt natuurlijk als het record is vergrendeld door een andere machine of een andere sessie.) Als gevolg hiervan kan een tweede machine een detailrecord dat niet is vergrendeld, misschien niet wijzigen als de bijbehorende hoofdtabel is vergrendeld, om redenen die hierboven zijn genoemd, hoewel detailrecords die niet zijn vergrendeld, wel kunnen worden verwijderd. Houd er ook rekening mee dat als u een record vergrendelt en vervolgens verwijdert, de vergrendeling ongedaan wordt gemaakt door de verwijdering. (En als ShowDeleted True is, blijft het record zichtbaar, hoewel het is ontgrendeld.)

## Automatische herhalingen instellen met setRetryPeriod

Zoals eerder gezegd, is het raadzaam na elke poging om een vergrendeling te plaatsen, te testen op de teruggegeven waarde van **lock**, tenzij u zeker weet dat de vergrendeling lukt. In bepaalde gevallen kunt u uit de besturingsstroom in de applicatie afleiden dat een vergrendeling uiteindelijk kan lukken als u lang genoeg blijft proberen de vergrendeling te plaatsen. In applicaties die bijvoorbeeld zijn ontworpen om 's nachts in een batchmodus te werken, kan het acceptabel zijn om enkele minuten te wachten tot een tabel beschikbaar wordt. In deze gevallen kunt u de besturingsstroom vaak vereenvoudigen met **setRetryPeriod** (Session-type).

Deze methode stelt een automatische herhalingsperiode in voor elke impliciete of expliciete bewerking die zou kunnen mislukken vanwege een resource-conflict. Dit is veel gemakkelijker dan code voor de herhaling toe te voegen aan uw methodes. Na de volgende instructie wordt bijvoorbeeld elke bewerking die zou kunnen mislukken vanwege een vergrendelde resource, automatisch gedurende honderd seconden herhaald:

```
setRetryPeriod(100)
```

***Opmerking*** Gebruik **setRetryPeriod** zorgvuldig. Uw applicatie moet nog steeds correct kunnen reageren als de herhalingsperiode zonder succes verstrijkt. De automatische herhalingsperiode wordt niet automatisch opnieuw ingesteld als de methode is afgelopen. Zorg er dus voor dat u de automatische herhaling uitschakelt als u deze niet meer nodig hebt:

```
setRetryPeriod(0)
```

Als **setRetryPeriod** blijft ingesteld op een groot getal, zal een gebruiker die later zonder succes een tabel of een record probeert te vergrendelen, in de war raken door de lange wachttijd. Het lijkt dan zelfs alsof Paradox verlamd is. U kunt de lengte van de huidige herhalingsperiode bepalen met **retryPeriod**.

## SetExclusive en setReadOnly

De Table-methode **setExclusive** werkt anders dan een volledige vergrendeling op een tabel. Als u **setExclusive** voor een Table-variabele aanroept voordat u een TCursor opent op de tabel, mislukt **setExclusive** als een andere gebruiker de tabel heeft geopend. Als de methode wel lukt, kan niemand anders de tabel meer openen.

Volledige vergrendelingen zijn krachtiger, omdat deze toestaan dat er veranderingen worden aangebracht in eigenschappen van de tabel. **setExclusive** daarentegen heeft alleen invloed op de tabulaire gegevens die worden beschreven door de Table-variabele.

**setExclusive** is nuttig voor applicaties die een alles-of-niets toegang tot een object nodig hebben. De gebruiker krijgt het exclusieve gebruik van het bestand, zolang de tabel open is. De plaatsing van een volledige vergrendeling met **lock** is echter in de meeste gevallen het geschiktst, omdat de applicatie een tabel lange tijd (of de hele tijd) niet-exclusief kan openhouden en vergrendelingen plaatst en vrijgeeft wanneer dat nodig is.

Met de Table-methode **setReadOnly**, die ook wordt aangeroepen voor een Table-variabele voordat er een TCursor op wordt geopend, kunt u ervoor zorgen dat de gebruiker van uw applicatie de tabel alleen kan lezen. Het informatiesysteem dat telefonistes gebruiken, is een goed voorbeeld van een applicatie waarbij het bijzonder nuttig is dat de gebruiker toegang heeft tot de informatie, zonder deze te kunnen wijzigen.

## Automatische opfrissing

Als Paradox een verandering in een tabel ontdekt (in het netwerk of gewoon omdat een ander venster of een andere TCursor in uw applicatie bepaalde gegevens wijzigt), wordt er een "opfrissing" uitgevoerd. Deze opfrissing vindt alleen plaats als de gegevens zijn gewijzigd die op dat moment op het scherm worden weergegeven. Er wordt geen opfrissing uitgevoerd voor wijzigingen van gegevens die niet op het scherm worden weergegeven.

Als dit gebeurt, start het formulier een DataRefresh-handeling. De standaardcode van het formulier voert de noodzakelijke bewerking uit zodat ObjectPAL de handeling kan voorbewerken of nabewerken. Deze handeling vindt pas plaats na de verandering. ObjectPAL kan dus niet meer doen dan interne berekeningen herstellen.

Het alleen-lezen kenmerk **Refresh** voor velden, records, tabelframes, multi-record objecten en formulieren is True vanaf het moment dat Paradox u vertelt dat er een opfrissing plaatsvindt tot het moment dat u alle bewerkingen hebt afgerond.

## Prestaties

U kunt volledige vergrendelingen en schrijfvergrendelingen zowel gebruiken om de prestaties te verbeteren als om de toegang te beperken. Als u bijvoorbeeld een schrijfvergrendeling plaatst op een gezamenlijk gebruikte tabel, worden veel bewerkingen sneller uitgevoerd, omdat Paradox niet hoeft te controleren of andere gebruikers de tabel hebben veranderd. Als u een volledige vergrendeling op een gezamenlijk gebruikte tabel plaatst, worden de bewerkingen nog sneller uitgevoerd, omdat Paradox de veranderingen die u in de tabel aanbrengt, in een geheugenbuffer kan opslaan, in plaats van alle veranderingen meteen weg te schrijven. Schrijfvergrendelingen en volledige vergrendelingen kunnen dus ook nuttig zijn als u niet verwacht dat andere gebruikers toegang willen tot de tabel.

HOOFDSTUK 19

# Fouten en foutverwerking

In dit hoofdstuk worden concepten en technieken beschreven die nuttig zijn bij de behandeling van runtime fouten in de ObjectPAL-omgeving. Dit hoofdstuk begint met een overzicht van de verschillende soorten fouten die zich voordoen als u ObjectPAL-applicaties maakt en uitvoert. Vervolgens worden de soorten runtime fouten gedefinieerd en wordt uitgelegd hoe ObjectPAL deze fouten onderverdeelt op grond van de ernst van de fout. Daarna worden de belangrijke concepten *foutstapel* en TRY...ONFAIL-blok geïntroduceerd. In dit hoofdstuk wordt vervolgens het standaardgedrag van het systeem bij fouten besproken en het hoofdstuk wordt afgesloten met informatie over het inbouwen van een eigen foutbehandeling in formulieren. De volgende onderwerpen worden in dit hoofdstuk besproken:

- Drie categorieën fouten: compiler-fouten, logische fouten en runtime fouten
- Fouten indelen naar de ernst van de fout
- De foutstapel
- Het TRY...ONFAIL-blok
- Standaardbehandeling van fouten door het systeem
- Eigen foutbehandeling
- Geavanceerde onderwerpen

## Foutcategorieën

Drie algemene foutcategorieën kunnen voorkomen dat een applicatie wordt uitgevoerd:

- Compiler-fouten
- Logische fouten

- Runtime fouten

## Compiler-fouten

*Compiler-fouten* zijn het resultaat van niet goed samengestelde instructies of van instructies die ObjectPAL niet herkent. Als de compiler een fout ontdekt, stopt deze en wordt een Editor-venster geopend voor de juiste codemodule (als dit nodig is). De cursor wordt geplaatst in de instructie die zich het dichtst bij de fout bevindt en er verschijnt een foutmelding op de statusregel van het Editor-venster. De bewerkmodus blijft actief, zodat u de fout onmiddellijk kunt zien en kunt corrigeren.

Compiler-fouten worden normaal gesproken veroorzaakt door verkeerd getypte, ontbrekende of verkeerd geplaatste elementen in uitdrukkingen. De compiler ontdekt bijvoorbeeld een onjuiste methode of een onjuiste aanroepreeks van een procedure als de lijst met parameters het verkeerde aantal of het verkeerde type argumenten bevat. De compiler ontdekt ook niet-overeenstemmende types als aan een variabele van een bepaald type een onjuiste waarde wordt toegewezen, bijvoorbeeld aan een String of een Smallint. In het volgende voorbeeld ziet u verschillende soorten fouten die de compiler ontdekt.

```
proc example (var hup smallInt)
   for i FORM 1 to 10     ; [1] sleutelwoord "FROM" verkeerd getypt
      hup = i
                          ; [2] FOR-lus vereist sleutelwoord "ENDFOR"
   hup = eigenproc()      ; [3] eigenproc() geeft een String terug, geen SmallInt
endProc

proc eigenproc() String
   return("een reeks" ; [4] sluithaakje ontbreekt
endProc
```

***Tip*** Het is verstandig alle variabelen op uw formulier expliciet te declareren. (Houd er rekening mee dat als u een variabele zonder declaratie gebruikt, deze variabele impliciet wordt gedeclareerd als AnyType.) Als u variabelen expliciet declareert, ontdekt de compiler bepaalde fouten die anders over het hoofd zouden worden gezien en wordt efficiëntere code gegenereerd.

## Logische fouten

*Logische fouten* zijn het resultaat van code die syntactisch correct is, maar onverwachte resultaten oplevert. Logische fouten zijn vaak het resultaat van verkeerde algoritmen of verkeerd gebruikte controle-structuren, zoals eindeloze lussen of berekeningen die de verkeerde resultaten teruggeven omdat deze op een onjuiste formule zijn gebaseerd. Een ander type logische fout heeft te maken met het bereik van variabelen, bijvoorbeeld als de variabele die u wilt bewerken, een andere variabele is met dezelfde naam.

Logische fouten kunnen soms runtime fouten veroorzaken (deze worden verderop beschreven). Een methode die zichzelf recursief aanroept kan bijvoorbeeld al het beschikbare geheugen of de hele stapel gebruiken en daardoor een runtime fout veroorzaken.

## Runtime fouten

*Runtime fouten* zijn het resultaat van instructies die kunnen worden gecompileerd (omdat ze syntactisch geldig zijn), maar om een bepaalde reden niet geheel kunnen worden uitgevoerd. Het klassieke voorbeeld is een uitdrukking waarin variabele A wordt gedeeld door variabele B en waarin variabele B de waarde 0 oplevert tijdens de uitvoering. Andere voorbeelden van bewerkingen die runtime fouten opleveren, zijn pogingen om een tabel te openen die niet bestaat, een diskette te lezen als de ingang van het station open staat, een onjuiste waarde toe te wijzen aan een variabele, of een kenmerk te manipuleren dat niet bij een bepaald object hoort.

De compiler onderschept duidelijke fouten in de verwijzing naar niet-bestaande kenmerken, maar kan een ongeldig kenmerk van een object niet ontdekken als het object bestaat uit een objectvariabele en niet uit een object met een specifieke naam. De volgende code is bijvoorbeeld syntactisch correct, maar veroorzaakt een fout als het object dat is toegewezen aan de UIObject-variabele *eenObject*, niet bestaat, of als het object bestaat, maar niet het kenmerk 'Color' heeft.

```
proc maakRood(var eenObject UIObject)
   eenObject.color = Red
endProc
```

Een ander type runtime fout ontstaat als Paradox geen resources meer heeft om de huidige bewerking uit te voeren. Een op hol geslagen recursieve **while**-lus, al genoemd in de bespreking van logische fouten, kan er bijvoorbeeld voor zorgen dat het systeem geen geheugenruimte of geen stapelruimte overhoudt.

## Samenvatting

Hoe meer ervaring u met ObjectPAL opdoet, hoe minder compiler-fouten uw code bevat. U kunt logische fouten reduceren door uw programma's zorgvuldig te plannen, te ontwerpen en te testen.

U dient echter vanaf het begin wel runtime foutbehandeling in uw code in te bouwen. (Hier volgt een voorbeeld: wat moet er gebeuren als de tabel waartoe u toegang wilt, onverwachts vergrendeld of verdwenen blijkt te zijn?) In de rest van dit hoofdstuk worden hulpmiddelen en technieken gepresenteerd voor de behandeling van runtime fouten in de ObjectPAL-omgeving.

# Runtime fouten begrijpen

In deze paragraaf worden verschillende kernbegrippen besproken die belangrijk zijn als u volledig gebruik wilt kunnen maken van de krachtige mogelijkheden die Paradox biedt voor foutbehandeling. Het gaat om de volgende onderwerpen:

- Niveaus van runtime fouten
- De foutstapel
- Het TRY...ONFAIL-blok

## Niveau van runtime fouten

ObjectPAL deelt runtime fouten in als *kritiek* of als *waarschuwing*, afhankelijk van de ernst van de fout.

Een *waarschuwingsfout* kunt u het best beschouwen als een mislukte terugkeer vanuit een aanroep van een methode of procedure, wat inhoudt dat de bewerking zonder succes is beëindigd. Een normale ObjectPAL-methode geeft de logische waarde True terug om een succesvolle beëindiging aan te geven en de logische waarde False om een waarschuwingsfout aan te geven.

Een *kritieke fout* ontstaat als Paradox vaststelt dat de vereiste bewerking niet succesvol kan worden afgesloten en dat het niet mogelijk of niet verstandig is om terug te keren met een waarschuwingsfout. Kritieke fouten kunnen worden veroorzaakt door:

- Een toewijzingsfout, zoals de verwijzing naar een niet-bestaand veld in een tabel
- De aanroep van een methode of procedure met ongeldige argumenten (als de aanroep niet wordt onderschept door de compiler)
- Een ongeldige teruggegeven waarde van een methode of procedure
- Een poging om voorbij het einde van een tabel of bestand te lezen

Als bijvoorbeeld de volgende instructie wordt uitgevoerd en *eenObject* niet het kenmerk 'Color' heeft, volgt er een kritische fout als u probeert een waarde aan het kenmerk 'Color' toe te wijzen.

```
eenObject.color = Red
```

### *Kritische of waarschuwingsfout?*

In een complexe omgeving als Paradox, die uit meerdere niveaus bestaat, is het soms moeilijk aan te geven welke foutsituaties kritische fouten veroorzaken en welke waarschuwingsfouten. In Paradox worden rekentaken verdeeld over een aantal modules (bijvoorbeeld de database-engine, het formuliersysteem, ObjectPAL enzovoort).

Deze modules communiceren onderling en kunnen op onvoorspelbare wijze fouten en waarschuwingen veroorzaken. In de volgende paragrafen wordt het verschil beschreven tussen de twee niveaus van runtime fouten. Deze beschrijving helpt u de juiste strategieën te ontwikkelen voor de behandeling van dergelijke fouten.

## Foutstapel

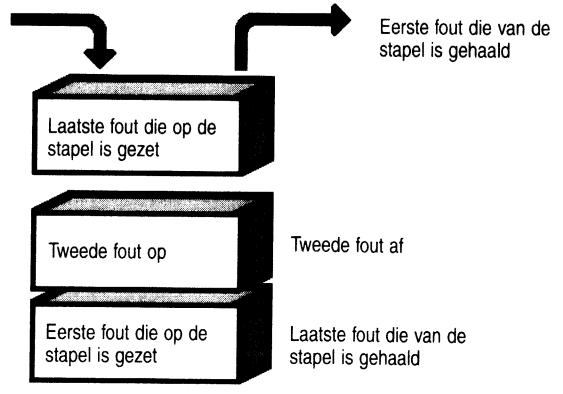
Paradox gebruikt een eenvoudig mechanisme, *foutstapel* genaamd, dat informatie teruggeeft over de runtime fout die het laatst is opgetreden. Telkens wanneer een runtime fout wordt ontdekt, slaat ObjectPAL de informatie over de fout op in de foutstapel. U kunt een aantal methodes gebruiken om deze informatie te onderzoeken en zelfs aan te passen.

Bijna elke ObjectPAL-instructie die een methode of procedure in de runtime bibliotheek aanroept, heeft invloed op de foutstapel. Uitzonderingen hierop zijn foutmethodes (bijvoorbeeld **errorCode**), methodes voor de weergave van berichten (bijvoorbeeld **msgInfo, message, beep**), en door de gebruiker gedefinieerde methodes en procedures. Als een instructie een runtime fout veroorzaakt, wordt de informatie over de fout op de stapel gezet. Als een instructie op een succesvolle manier wordt uitgevoerd, wordt de stapel leeggemaakt. Als u dit weet, is de volgende logische vraag: "Wat is dit voor informatie en hoe kan ik deze informatie gebruiken?" In de volgende paragrafen vindt u de antwoorden op deze vraag.

### *Informatie in de foutstapel*

Zoals u ziet in Afbeelding 19-1, werkt het mechanisme voor de foutrapportage als een LIFO-stapel (Last In First Out: Eerste erin Laatste eruit). De informatie die het laatst op de stapel wordt gezet, wordt er dus het eerst afgehaald.

Afbeelding 19-1 De foutstapel



Telkens wanneer zich een runtime fout voordoet, zet Paradox een of meer records met foutinformatie op de foutstapel. Elk record bevat:

- Een *foutcode*, een numerieke waarde die aangeeft om welke fout het gaat
- Een *foutmelding*, een tekstreeks die aan de gebruiker kan worden getoond

**Belangrijk** Voordat een methode of procedure wordt uitgevoerd, wordt de stapel leeggemaakt. Daarom is alleen informatie beschikbaar over de laatst uitgevoerde methode. Wat is nu het nut van de stapel?

Eén ObjectPAL-instructie kan verschillende lagen foutinformatie genereren wanneer de verschillende Paradox-modules de fout tegenkomen en foutinformatie op de stapel zetten.

Stel dat een eigen methode probeert een TCursor te openen op een niet-bestaande tabel. Dit zou fouten veroorzaken in twee verschillende gebieden: in de database-engine-module en in de runtime module van ObjectPAL. De database-engine zou de fout het eerst ontdekken en zou foutinformatie van een laag niveau op de foutstapel zetten. Vervolgens ontdekt de runtime module de fout en zet deze informatie vanaf het eigen niveau op de stapel. In dit voorbeeld bevat de stapel twee berichten: het bovenste element van de stapel, "**Er is een fout veroorzaakt in de methode 'open' op een object van het type TCursor**", gevolgd door een tweede element, "**Tabel bestaat niet. Bestand hoi.db.**"

## Procedures foutstapel

ObjectPAL kent verschillende methodes (gedefinieerd voor het System-type) voor het onderzoeken, weergeven en veranderen van informatie over runtime fouten. U gebruikt deze procedures om te bepalen wat de fout was, om te bepalen waarom de fout plaatsvond, om informatie over de fout aan de stapel toe te voegen of in de stapel te wijzigen en om deze informatie aan de gebruiker te tonen.

**errorCode** en **errorMessage** geven respectievelijk de foutcode en de foutmelding terug vanaf de top van de stapel. Omdat de stapel door deze procedures ongewijzigd blijft, is **errorPop** beschikbaar om één element tegelijk van de stapel te verwijderen en zo toegang te geven tot de resterende elementen. De **errorClear**-methode verwijdert alle elementen uit de stapel. De **errorLog**-procedure zet een nieuw element (foutcode plus berichtreeks) op de foutstapel. Ten slotte roept **errorShow** het standaard foutdialoogvenster van ObjectPAL aan, dat de gebruiker een standaarddialoogformulier toont waarmee door de foutstapel kan worden gebladerd.

Bij wijze van voorbeeld toont deze eenvoudige methode voor foutwaarschuwing de foutcode op de titelbalk van het dialoogvenster en de foutmelding in het kader zelf.

```
proc mijnFoutmelding()
   msgInfo("foutcode: " + errorCode(), errorMessage())
endProc
```

***Opmerking*** Als u **errorCode** en **errorMessage** aanroept, heeft dit geen invloed op de inhoud van de foutstapel. Andere methodes en procedures veranderen de foutstapel wel. Roep dus eerst deze procedures aan in uw routines voor foutbehandeling.

## *Het foutdialoogvenster weergeven*

Als u foutinformatie wilt weergeven, kunt u **errorShow** gebruiken om het standaard foutdialoogvenster te openen. Dit dialoogvenster lijkt veel op het dialoogvenster dat door het runtime systeem van het formulier wordt gebruikt. De gebruiker kan met dit dialoogvenster door de foutstapel bladeren.

***Opmerking*** **errorShow** verwijdert alle elementen uit de foutstapel. Wees er dus zeker van dat u deze informatie niet meer nodig hebt, voordat u deze procedure aanroept.

In een afgeronde applicatie is het waarschijnlijk verstandig meer dan alleen de foutinformatie weer te geven. U kunt bijvoorbeeld bepaalde types fouten zelf behandelen en Paradox de andere fouten laten behandelen. In de volgende paragrafen ziet u hoe u de foutstapel-procedures gebruikt door een eigen procedure te maken voor foutbehandeling.

## *errorCode*

Het voorbeeld begint met het gebruik van **errorCode** in een **switch... endSwitch**-structuur om bepaalde foutcodes op te vangen en te behandelen. Fouten zijn voor de duidelijkheid gecodeerd met behulp van standaardfoutconstanten van Paradox. **errorCode** geeft de foutcode terug van boven op de foutstapel. U gebruikt een **switch**-instructie om codes buiten een bepaalde verzameling foutcodes uit te sluiten.

```
proc onzeFoutProc()
var
   foutc smallInt
endVar
   foutc = errorCode()
   switch                          ; alleen bepaalde foutvoorwaarden behandelen
     case fout = peObjectNotFound    : noObject()   ; voer een eigen methode uit
     case foutc = pePropertyNotFound : noProperty() ; voer een eigen methode uit
     otherwise                       : return
   endSwitch
   ;meer verwerking...
endProc
```

ObjectPAL kent constanten voor foutcodes. Als u de lijst wilt zien, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Constanten'. Kies vervolgens 'Errors' in de kolom 'Types constanten'. De constanten verschijnen in de kolom 'Constanten'. De online ObjectPAL Help bevat ook een lijst met constanten. U kunt deze

### errorMessage

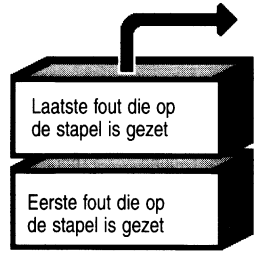
**errorMessage** geeft de tekst van de foutmelding van boven op de foutstapel terug. **errorMessage** wordt veel gebruikt om foutmeldingen bij te houden in een bestand. U kunt deze procedure ook gebruiken om de standaardfoutmelding te combineren met een eigen bericht. In de volgende code ziet u beide mogelijkheden. Dit voorbeeld voegt een eigen routine toe aan het vorige voorbeeld om de fouten te noteren.

```
proc onzeFoutProc()
var
   foutc smallInt
   fouts String
   foutMeld TextStream
endVar
   foutc = errorCode()
   switch                          ; alleen bepaalde foutvoorwaarden behandelen
      case foutc = peObjectNotFound   : geenObject()  ; voer een eigen methode
                                                      ; uit
      case foutc = pePropertyNotFound : geenKenmerk() ; voer een eigen methode
                                                      ; uit
      otherwise                       : return
   endSwitch
   fouts = String(Date())+","+String(errorCode())+","+errorMessage()
   foutMeld.open("foutLog.txt","A")
   foutMeld.writeLine(fouts)
   foutMeld.close()
   ;meer eigen verwerking...
endProc
```

### errorPop

U gebruikt **errorPop** om één laag tegelijk van de foutstapel af te halen zodat volgende niveaus van foutinformatie beschikbaar worden. (Zie afbeelding 19-2.) (Als u alle lagen van de stapel wilt verwijderen, gebruikt u **errorClear**.)

Afbeelding 19-2 De foutstapel: informatie erop zetten of eraf halen




Als u **errorShow, errorCode, errorMessage** of **errorLevel** aanroept, krijgt u informatie van boven op de stapel.

Als u informatie van een lager niveau wilt ophalen, roept u **errorPop** aan en vervolgens **errorShow, errorCode, errorMessage** of **errorLevel**.

**errorPop** geeft True terug als de procedure succesvol is geweest en False als dit niet zo is (omdat de stapel geen niveaus meer bevat). In het volgende voorbeeld wordt de hele foutstapel in een bestand genoteerd.

```
proc onzeFoutProc()
var
```

het volgende voorbeeld wordt de hele foutstapel in een bestand genoteerd.

```
proc onzeFoutProc()
var
   foutc smallInt
   fouts String
   foutMeld TextStream
endVar
   foutc = errorCode()
   switch                          ; alleen bepaalde foutvoorwaarden behandelen
      case foutc = peObjectNotFound   : geenObject()  ; voer een eigen methode
                                                      ; uit
      case foutc = pePropertyNotFound : geenKenmerk() ; voer een eigen methode
                                                      ; uit
      otherwise                       : return
   endSwitch
   fouts = String(Date())+","+String(errorCode())+","+errorMessage()
   foutMeld.open("foutLog.txt","A")
   foutMeld.writeLine(fouts)
   While errorPop()
      fouts = String(Date())+","+String(errorCode())+","+errorMessage()
      foutMeld.writeLine(fouts)
   endWhile
   foutMeld.close()
   ;meer eigen verwerking...
endProc
```

### *Informatie toevoegen aan de stapel*

U gebruikt **errorLog** om informatie aan te passen in de foutstapel of toe te voegen aan de foutstapel. **errorLog** heeft twee argumenten, een foutcode en een foutmelding. In dit voorbeeld wordt een eigen waarschuwing op de foutstapel gezet voor fouten die in deze routine niet worden behandeld.

```
proc onzeFoutProc()
var
   foutc smallInt
   fouts String
   foutMeld TextStream
endVar
   foutc = errorCode()
   switch                          ; alleen bepaalde foutvoorwaarden behandelen
      case foutc = peObjectNotFound   : geenObject()  ; voer eigen methode uit
      case foutc = pePropertyNotFound : geenKenmerk() ; voer eigen methode uit
      otherwise                       :
         errorLog(MYCODE, "Kan deze niet behandelen")
         geavanceerdeFoutProc()  ; roep meer gespecialiseerde verwerking aan
         return
   endSwitch
   fouts = String(Date())+","+String(errorCode())+","+errorMessage
   foutMeld.open("foutLog.txt","A")
   foutMeld.writeLine(fouts)
   While errorPop()
      fouts = String(Date())+","+String(errorCode())+","+errorMessage()
      foutMeld.writeLine(fouts)
   endWhile
   foutMeld.close()
   ;meer eigen verwerking...
endProc
```

## TRY...ONFAIL-blok

Het TRY...ONFAIL-blok bestaat uit een *transactieblok*, dat een of meer ObjectPAL-instructies bevat (een *transactie* genoemd), en een *herstelblok*, dat ook bestaat uit één of meer ObjectPAL-instructies die aangeven wat er moet gebeuren als de transactie mislukt. Oftewel, u wilt dat de *transactie* slaagt en als dit niet het geval is, wilt u dat de besturing wordt doorgegeven aan het *herstelblok*. Het TRY...ONFAIL-blok ziet er als volgt uit:

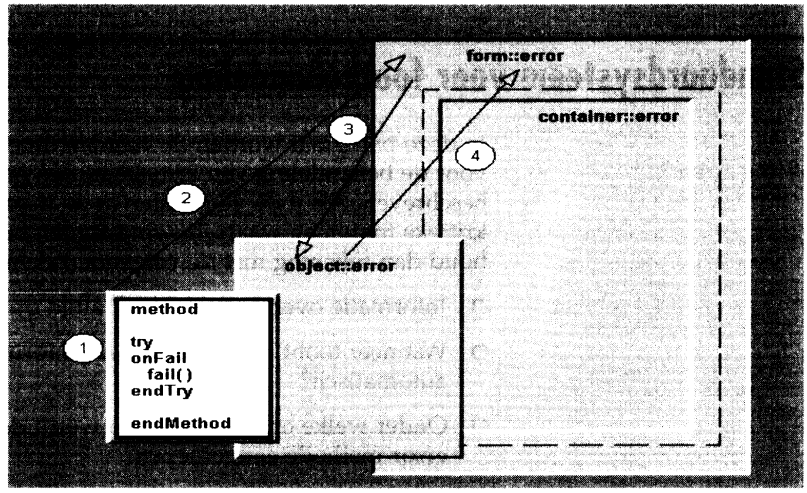
```
TRY
   [transactieblok]
ONFAIL
   [
      herstelblok
      [RETRY]                          ; optioneel sleutelwoord
      [FAIL ( [ foutCode, foutMeld] ) ]  ; optionele procedure, optionele
argumenten
   ]
ENDTRY
```

Als de transactie slaagt, springt het programma naar het sleutelwoord ENDTRY en wordt de foutstapel leeggemaakt. Als een instructie binnen de transactie mislukt, gaat de besturing onmiddellijk naar het herstelblok.

Als bijvoorbeeld de vierde instructie in een transactie van vijf instructies mislukt, gaat de uitvoering door bij de eerste instructie van het ONFAIL-blok. De vijfde instructie van de transactie wordt niet uitgevoerd, maar de effecten van de eerste drie instructies, die met succes werden uitgevoerd, blijven bestaan.

***Belangrijk*** U kunt een of meer expliciete TRY...ONFAIL-blokken nesten rond kritieke codegedeelten. Als een transactieblok mislukt, borrelt de fout naar het volgende blok enzovoort, totdat zich geen blokken meer in de methode bevinden. Op dat moment gaat de besturing naar de herstelclausule in het impliciete TRY...ONFAIL-blok dat Paradox rond iedere methode plaatst. Vanaf dit punt wordt een ErrorEvent gegenereerd en naar het formulier gezonden, zoals u ziet in Afbeelding 19-3.

Afbeelding 19-3 Het TRY...ONFAIL-model




1. Binnen een methode probeert een TRY...ONFAIL-blok een transactie tot stand te brengen. In het herstelblok genereert een aanroep van **fail** een ErrorEvent.

2. De fout activeert de ingebouwde **error**-methode van het formulier.

3. De **error**-methode roept standaard de **error**-methode aan van het object waarvan de code de fout heeft veroorzaakt.

4. De fout borrelt standaard omhoog in de hiërarchie van ingesloten objecten totdat deze de **error**-methode van het formulier bereikt, waardoor een dialoogvenster wordt geopend.

### *retry*

Binnen het herstelblok kunt u het sleutelwoord **retry** gebruiken om het transactieblok opnieuw te laten uitvoeren.

### *fail-procedure*

U kunt ook een mislukking forceren door de **fail**-procedure van het System-type aan te roepen in het herstelblok of in procedures die worden aangeroepen door het herstelblok. **fail** kan worden aangeroepen zonder argumenten (de haakjes zijn dan echter nog steeds nodig) of met een foutcode en een foutmelding. Bijvoorbeeld:

```
fail(peObjectNotFound, "Het object bestaat niet!")
```

Een **fail**-procedure kan worden genest in een of meer aanroepen van eigen methodes of procedures vanuit het **onFail**-blok.

***Opmerking*** Wees in deze gevallen voorzichtig met het gebruik van **fail**. Als variabelen die lokaal zijn ten opzichte van alle geneste methodes of procedures, van de stapel worden verwijderd, wordt de toewijzing van speciale objecten (zoals grote tekstblokken) ongedaan gemaakt en worden lokale verwijzingsobjecten (zoals TCursors) gesloten. Natuurlijk blijven veranderingen van variabelen buiten het bereik van deze methodes of procedures intact, evenals veranderingen in

tabellen die met succes zijn aangebracht voordat de mislukking zich voordeed.

# Standaardsysteem voor foutbehandeling

In deze paragraaf worden de standaardmechanismen van Paradox voor de behandeling van kritieke fouten en waarschuwingsfouten beschreven. Als u de verschillen in de manier waarop ObjectPAL kritieke fouten en waarschuwingsfouten behandelt, wilt begrijpen, houd dan rekening met de volgende zaken als u deze paragraaf leest:

- Informatie over de fout wordt altijd op de foutstapel gezet.
- Wanneer toont het systeem deze informatie standaard automatisch?
- Onder welke omstandigheden wordt de ingebouwde **error**-methode aangeroepen?
- Waar wordt uitvoering van uw code hervat na een fout?
- Kan het standaardfoutgedrag van het systeem worden aangepast?

Fouten kunnen verder worden onderverdeeld in twee categorieën: fouten die zijn gegenereerd als gevolg van een activiteit van ObjectPAL en fouten die zijn gegenereerd als gevolg van de interactie van de gebruiker met Paradox. In deze paragraaf wordt de eerste categorie behandeld. De laatste categorie wordt behandeld in de paragraaf "Geavanceerde aspecten van foutbehandeling", aan het einde van dit hoofdstuk.

## Waarschuwingsfouten

Een waarschuwingsfout wordt gekenmerkt door het volgende standaardgedrag van het systeem:

- Paradox zet informatie over de fout op de foutstapel.
- De programmabesturing wordt teruggegeven aan de volgende instructie in uw code. U merkt de fout op door de teruggegeven waarde van de mislukte methode of procedure te controleren (standaard programmeergebruik).
- Stapelinformatie over de fout wordt *niet* standaard getoond, maar u kunt deze onderzoeken met behulp van de foutmethodes.
- De ingebouwde **error**-methode wordt *niet* standaard aangeroepen, maar kan wel worden aangeroepen met **fail**.
- U kunt het standaardgedrag veranderen. Zoals u in een volgende paragraaf zult zien, kunt u het systeem opdragen waarschuwingsfouten hetzelfde te behandelen als kritieke fouten.

## Kritieke fouten

Kritieke fouten worden gekenmerkt door het volgende standaardgedrag van het systeem:

- Paradox zet informatie over de fout op de foutstapel.
- De besturing wordt teruggegeven aan het ONFAIL-gedeelte van het impliciete TRY...ONFAIL-blok. (Een uitleg volgt.)
- De ingebouwde **error**-methode wordt aangeroepen.
- Als de fout naar het niveau van het formulier borrelt, wordt stapelinformatie over de fout weergegeven door middel van het standaarddialoogvenster voor fouten van Paradox.
- U kunt het standaardgedrag aanpassen door eigen foutmethodes te maken of door een expliciet TRY...ONFAIL-blok te plaatsen rond de juiste instructies of blokken instructies met logica voor foutbehandeling.

### *Impliciet TRY...ONFAIL-blok*

Zoals u een TRY...ONFAIL-blok in uw ObjectPAL-code rond een transactie kunt plaatsen, zo plaatst Paradox een impliciet TRY...ONFAIL-blok rond elke ingebouwde methode voor elk ontwerpobject op een formulier. Met andere woorden, Paradox behandelt elke ingebouwde methode als een aparte transactie.

Als een instructie van ObjectPAL een runtime fout veroorzaakt, mislukt de transactie van de ingebouwde methode. Paradox kijkt eerst naar de code om te bepalen of de instructie zich in een expliciet TRY...ONFAIL-blok bevindt. Als er niet een dergelijk blok is, als het blok de fout niet op een adequate manier behandelt of als het blok **fail** aanroept, gaat de besturing onmiddellijk naar het *impliciete* TRY...ONFAIL-blok, dat vervolgens een ErrorEvent genereert. Met deze ErrorEvent wordt de juiste **error**-methode aangeroepen.

### *Ingebouwde methode error*

Elk UIObject heeft een ingebouwde methode met de naam **error**, die wordt uitgevoerd als een fout niet geheel in het ONFAIL-blok wordt behandeld of als een ObjectPAL-instructie de **fail**-methode aanroept.

Volgens het standaardgedrag van Paradox gaat de ErrorEvent eerst naar het formulier. Daarna roept het formulier de ingebouwde **error**-methode aan van het huidige object, waarvan de code de fout heeft veroorzaakt. Het huidige object kan de fout behandelen, de fout omhoog laten borrelen door de hiërarchie van ingesloten objecten of kan beide dingen doen.

*Het actiemodel wordt beschreven in Hoofdstuk 12.*

Stel dat een formulier een tabelframe bevat en dat code die is gekoppeld aan het tabelframe, een kritieke fout veroorzaakt. Als die code wordt uitgevoerd en de fout veroorzaakt, wordt een ErrorEvent geactiveerd die de ingebouwde **error**-methode van het formulier activeert. De ingebouwde **error**-methode van het formulier verzendt

de ErrorEvent naar de ingebouwde **error**-methode van het tabelframe, vanwaar deze eventueel omhoogborrelt via de hiërarchie van ingesloten objecten, totdat de fout het formulier bereikt. Ten slotte geeft de ingebouwde **error**-methode van het formulier een dialoogvenster weer om u in te lichten over de fout.

# Eigen foutbehandeling

In deze paragraaf worden voorbeelden getoond van de basistechnieken die u kunt gebruiken voor de behandeling van waarschuwingsfouten en kritieke runtime fouten.

## Waarschuwingsfouten behandelen

De algemene werkwijze voor de behandeling van waarschuwingsfouten is bij de meeste programmeurs bekend: als er een kans op mislukking bestaat in de uitvoeringsomgeving, test dan de teruggegeven waarde van methodes en procedures op succes of mislukking.

Aangezien waarschuwingsfouten meestal worden veroorzaakt door methodes die een logische waarde teruggeven, kunt u deze behandelen binnen een **if...then**-blok of een **switch**-instructie. De teruggegeven waarden van de methodes **attach** en **rename** van het Table-type in het volgende voorbeeld worden met behulp van een **if...then**-blok gecontroleerd op succes of mislukking.

```
proc eenProc()
   var
      t Table
   endVar
   if not t.attach("oudenaam.db") then
      msgInfo("Waarschuwing", "Kan de tabel niet openen.") ; meld de fout
      openError() ; voer eigen code uit om de fout te behandelen
      return
   endif
   if not t.rename("nwenaam.db") then
      renameError() ; voer eigen code uit om de fout te behandelen
      msgInfo("Waarschuwing", "Kan de tabel niet hernoemen.") ; meld de fout
      return
   endIf
endProc
```

U gebruikt **errorCode** en **errorMessage** in combinatie met de eerder besproken filtermethodes om vanuit de foutstapel foutinformatie over de waarschuwing te krijgen of weer te geven, zoals in het volgende voorbeeld.

```
proc eenProc()
var eigenTC TCursor endVar
      if eigenTC.close() then  ; equivalent van "if eigenTC.close() = True then"
      doEIets()           ; verwerking
   else
      if errorCode() = peTableClose then
         msgInfo("Probleem", errorMessage()) ; meld de fout aan de gebruiker
```

```
      else
         ; roep een eigen procedure aan
         meerFoutVerwerking(errorCode(),"Verdere verwerking nodig")
      endIf
   endIf
endProc
```

## Kritieke fouten behandelen

De twee belangrijkste benaderingen voor de behandeling van kritieke runtime fouten zijn het **try...onfail**-blok en de aangepaste ingebouwde **error-methode**.

### *TRY...ONFAIL-blok*

In de eerste benadering wordt gebruik gemaakt van het feit dat ObjectPAL de besturing na een kritieke fout teruggeeft aan de eerste instructie in het FAIL-blok van het binnenste TRY...ONFAIL-blok. In het volgende voorbeeld wordt het TRY...ONFAIL-blok gebruikt om kritieke fouten te onderscheppen die te maken hebben met de instelling van een kenmerk van een niet-bestaand object. Stel dat een formulier *kader1* insluit en dat *kader1 kader2* insluit.

```
method pushButton(var eventInfo Event)
var s String endVar

kader1.kader2.color = Blue                    ; dit werkt
s = "kader5"                                  ; kader5 bestaat niet

try
   kader1.(s).color = Red                     ; probeer kleur van kader5 in te
                                              ; stellen
onFail                                        ; behandel de fout
   msgStop("Fout", "Kan niet vinden " + s)
   s = "kader2"                               ; kader2 bestaat
   reTry                                      ; probeer opnieuw
endTry

s = "kader6"                                  ; kader6 bestaat niet
try
   kader1.(s).color = Green
onFail
   fail(peObjectNotFound, "Het object " + s + " bestaat niet.")
endTry
endMethod
```

In het volgende voorbeeld ontstaat een overloopfout als een getal dat buiten het bereik ligt, wordt toegewezen aan een variabele van het type SmallInt. **retry** en **fail** worden gebruikt om op verschillende wijze op bepaalde foutcodes te reageren.

```
method pushButton(var eventInfo Event)
var sm    smallInt
    maxsm smallInt
endVar
   maxsm = 32767
   try
      sm = maxsm + 1
   onFail
      if errorCode() = peOverFlow then
         msgStop("Fout", "overloopfout bij toewijzing")
         maxsm =-10
         retry
```

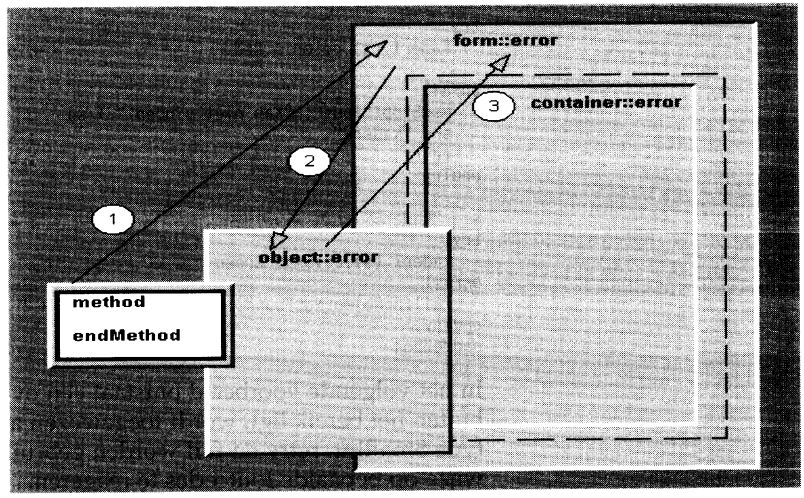
```
      else
         fail()
      endif
   endTry
endMethod
```

***Opmerking*** De foutcode **peBreak** wordt gegenereerd als een gebruiker een programma-activiteit onderbreekt door op *Ctrl-Break* te drukken. Afhankelijk van de eisen die uw applicatie stelt, kan het daarom erg nuttig zijn in het ONFAIL-blok van kritieke bewerkingen te testen op de **peBreak**-foutcode.

## *error*

Zoals u in Afbeelding 19-4 ziet, kunt u objectspecifieke code voor de behandeling van fouten in de ingebouwde **error**-methode van het object plaatsen. U kunt ook fouten behandelen voor groepen objecten door code te plaatsen in de ingebouwde **error**-methode van een object in de hiërarchie van ingesloten objecten. Ten slotte kunt u fouten voor alle objecten op het formulier op een globale manier behandelen door code te koppelen aan de ingebouwde **error**-methode van het formulier.

Afbeelding 19-4 Het actiemodel voor ErrorEvents




1. Een instructie in een methode veroorzaakt een fout, die de **error**-methode van het formulier activeert.

2. Het formulier roept standaard de **error**-methode aan van het object waarvan de code de fout heeft veroorzaakt.

3. De fout borrelt standaard via de hiërarchie van ingesloten objecten terug naar de **error**-methode van het formulier; deze verricht de juiste handeling.

Als de fout het formulier voor de tweede keer bereikt, toont het formulier standaard een dialoogvenster.

Informatie over het foutniveau wordt opgeslagen in het actiepakket *eventInfo*. Als u eigen code schrijft voor de **error**-methode, maak dan respectievelijk gebruik van de methodes **reason** en **setReason** van het ErrorEvent-type, om deze informatie te lezen en te schrijven. U kunt de ernst van de fout bepalen met de ObjectPAL-constanten ErrorCritical en ErrorWarning. Deze constanten zijn gedefinieerd voor gebruik met **reason** en **setReason**.

Het nut van deze constanten en van de kracht van de **error**-methode wordt duidelijk als u zich voorstelt dat u alle fouten wilt behandelen als kritieke fouten. (De methode **trapOnWarning**, die verderop in dit hoofdstuk wordt behandeld, doet dit trouwens automatisch, maar dit voorbeeld is de moeite waard.) U bereikt het gewenste effect met de volgende code die is gekoppeld aan de ingebouwde **error**-methode van een object:

```
method error(var eventInfo ErrorEvent)
   if eventInfo.reason() = ErrorWarning then
      eventInfo.setReason(ErrorCritical)
   endIf
endMethod
```

Als de **error**-methode van dit object wordt uitgevoerd, roept de methode **reason** aan om het foutniveau te bepalen. Vervolgens wordt de fout, indien nodig, naar een kritieke fout geconverteerd en borrelt het actiepakket naar het formulier via de hiërarchie van ingesloten objecten. Het formulier toont dan standaard het dialoogvenster voor kritieke fouten.

# Geavanceerde aspecten van foutbehandeling

In deze paragraaf worden de volgende onderwerpen behandeld:

- Interactieve fouten
- onWarnings onderscheppen
- Waar moet code worden gekoppeld?

## Interactieve fouten

In deze paragraaf wordt de standaardreactie van het systeem beschreven die optreedt bij fouten die niet zijn gegenereerd door ObjectPAL-activiteit, maar het gevolg zijn van interactie met de gebruiker. Als u bijvoorbeeld het toetsenbord gebruikt om door een tabelframe te schuiven en probeert voorbij het einde van de tabel te schuiven, veroorzaakt dit een interactieve fout. Als zich een interactieve fout voordoet, gebeurt het volgende:

- Paradox genereert een ErrorEvent en de foutmethode van het actieve object wordt aangeroepen. Zoals bij elke ingebouwde methode wordt het formulier het eerst aangeroepen. Vervolgens

gaat de actie naar het actieve object (de 'schuldige' die de fout heeft gegenereerd in dit geval). Als de actie hier niet wordt behandeld, borrelt deze omhoog in de hiërarchie van ingesloten objecten, totdat het formulier weer wordt bereikt.

- Als de ErrorEvent inderdaad voor de tweede keer het formulier bereikt, en als de fout een interactieve *waarschuwingsfout* was, genereert de ingebouwde **error**-methode van het formulier een StatusEvent. Deze roept de ingebouwde **status**-methode aan, die een soortgelijke route volgt door het formulier, naar het actieve object, en terug naar het formulier, waarna een statusbericht verschijnt in het statusberichtgebied van het scherm.
- Als de ErrorEvent de foutmethode van het formulier voor een tweede keer bereikt, maar de fout een interactieve *kritieke* fout was, toont de ingebouwde **error**-methode het standaard-dialoogformulier voor fouten van Paradox.

### *Standaardgedrag veranderen*

Hoewel u geen ObjectPAL-code kunt schrijven om de standaard-behandeling van interactieve waarschuwingsfouten te vervangen, kunt u de StatusEvent onderscheppen en eigen code plaatsen in de **status**-methode van een object of op het formulier. U kunt bijvoorbeeld de waarschuwingsberichten aanpassen of deze noteren in een logboek. Denk eraan dat de StatusEvent eerst naar de ingebouwde **status**-methode gaat, die deze vervolgens verzendt naar de **status**-methode van het actieve object (het object dat de fout heeft veroorzaakt). Een StatusEvent borrelt standaard via de hiërarchie van ingesloten objecten omhoog totdat het formulier weer wordt bereikt. Zie Hoofdstuk 12 voor meer informatie over StatusEvents.

## Waarschuwingsfouten en TRY...ONFAIL

Een TRY...FAIL-blok onderschept standaard geen waarschuwings-fouten De **ErrorTrapOnWarnings**-methode van het Session-type, die als argument de logische waarde TRUE of FALSE heeft, kan ObjectPAL opdragen waarschuwingsfouten in een TRY...FAIL-blok te onderscheppen, zoals dit ook gebeurt met kritieke fouten. De standaardbewerking is dan hetzelfde als voor kritieke fouten, maar het eindresultaat is een bericht op de statusbalk in plaats van het standaarddialoogvenster voor fouten van Paradox.

U kunt **ErrorTrapOnWarnings**-methodes tijdens een sessie altijd en zo vaak u wilt in- of uitschakelen. Stel dat u bepaalde waarschuwings-fouten binnen TRY...FAIL-blokken wilt behandelen en andere waar-schuwingsfouten binnen IF...THEN-blokken. U neemt dan al naar gelang de omstandigheden een van de volgende instructies op in de routine voor de foutbehandeling:

```
ErrorTrapOnWarnings(Yes)
```

Of:

```
ErrorTrapOnWarnings(No)
```

## Waar moet de code worden gekoppeld?

Het actiemodel van ObjectPAL biedt veel flexibiliteit bij het werken met de foutstapel en de ingebouwde **error**-methode. U moet echter aan een aantal zaken denken als u beslist waar u uw code voor foutbehandeling koppelt. U moet misschien een compromis zoeken, zoals u ziet in de volgende paragrafen.

Denk eraan dat alle acties, inclusief ErrorEvents, eerst naar het formulier gaan. Dit betekent dat u fouten centraal kunt behandelen door code te koppelen aan de ingebouwde **error**-methode van het formulier. In enkele applicaties kan dit verstandig zijn, maar het gaat soms ten koste van andere zaken.

*Complexiteit*

Misschien wilt u niet elke fout op het niveau van het formulier behandelen, omdat dit te *complex* wordt. Als u fouten voor elk object en van elk type op een formulier van een willekeurige grootte behandelt, kan dit erg ingewikkeld worden. Stel dat u een formulier hebt dat een aantal objecten bevat: knoppen, kaders, tekstvakken, bitmaps en een tabelframe. In dit formulier kan slechts één object, het tabelframe, een inbreuk op een sleutel genereren. Het is dan wellicht verstandiger de foutbehandeling plaatselijk te houden en code te koppelen aan de **error**-methode van het tabelframe, zoals u in het volgende voorbeeld ziet:

```
method error (var eventInfo ErrorEvent)
   if errorCode() = peKeyViolation then
      behandel()   ; roep eigen foutbehandeling aan
   endIf
endMethod
```

*Overdraagbaarheid*

*Overdraagbaarheid* is een tweede onderwerp waarmee u rekening moet houden als het gaat om globale foutbehandeling op formulierniveau. Het is veel gemakkelijker objecten en groepen objecten tussen formulieren en applicaties te knippen en te plakken als de foutbehandeling plaatselijk bij de objecten wordt uitgevoerd.

Een techniek die vaak van pas komt is *groepering*. Als u een grote groep veldobjecten hebt, kunt u code koppelen aan de ingebouwde **error**-methode van elk veldobject. Zo krijgt elk veldobject eigen foutbehandeling. U kunt de velden echter ook in een insluitend object plaatsen (zoals een kader) en de eigen **error**-methode koppelen aan het kader. Op deze manier wordt een fout in een van de veldobjecten behandeld door de **error**-methode van het kader.

Er zijn geen algemeen geldende regels voor de beste plaats voor foutbehandeling. U kunt de methodes die in deze paragraaf zijn besproken, in elke combinatie gebruiken en code koppelen aan het formulier, aan de insluitende kaders en zelfs aan een of meer velden en andere objecten.

HOOFDSTUK 20

# Scripts maken en afspelen

Een script bestaat uit ObjectPAL-code in een eigen bestand, die niet is gekoppeld aan een formulier. Een script is een object en verschijnt als pictogram op het bureaublad, maar het heeft geen type, verschijnt niet in een venster en bevat geen ontwerpobjecten. Een script heeft de ingebouwde methodes **run**, **error** en **status** die u kunt uitvoeren als u Paradox interactief gebruikt of kunt aanroepen vanuit een ObjectPAL-methode of -procedure. Zoals elk ander object heeft een script ook vensters voor de declaratie van variabelen, constanten, procedures, types en externe routines. U kunt ook eigen methodes declareren.

U gebruikt een script als u code wilt uitvoeren zonder een formuliervenster te openen en weer te geven.U kunt bijvoorbeeld scripts gebruiken om een **scan...endScan**-lus uit te voeren om de waarden van een veld om te zetten in hoofdletters of om een overzicht te maken van de kenmerken en methodes die aan een formulier zijn gekoppeld. Deze taken kunt u ook verrichten door een leeg formulier te maken, er een knop in te plaatsen en vervolgens de juiste code te koppelen aan de **pushButton**-methode, maar een script slaat de eerste twee stappen over.

Vanuit een script hebt u volledig toegang tot de runtime bibliotheek van ObjectPAL, zodat u andere objecten kunt besturen. U kunt bijvoorbeeld andere scripts aanroepen, tabellen, formulieren en rapporten openen en gebruiken en queries uitvoeren. U kunt methodes aanroepen die zijn gekoppeld aan andere objecten, en de kenmerken van deze objecten opvragen en instellen.

# Script maken

Als u een script wilt maken, gaat u als volgt te werk:

1. Kies 'Bestand | Nieuw | Script'. Er wordt een venster van de ObjectPAL-Editor geopend dat de volgende tekst bevat:

   ```
   method run(var eventInfo Event)

   endMethod
   ```

2. Dit is een standaardvenster van de ObjectPAL-Editor. U kunt dus de **run**-methode bewerken, de syntaxis van de methode controleren en fouten in de methode opsporen, zoals bij elke andere methode. Net als vanuit andere objecten kunt u vanuit een script andere formulieren openen en sluiten, objecten maken, kenmerken en waarden opvragen en instellen, berichten weergeven en methodes activeren.

3. U opent het methodevenster door 'Taal | Methodes' te kiezen. In dit venster kunt u variabelen, constanten, gegevenstypes, procedures, eigen methodes en DLL's declareren die u wilt gebruiken. Denk er echter aan dat alles wat u declareert alleen zichtbaar is voor de **run**-methode van het script.

4. Als u klaar bent met de bewerking, sluit u het venster. Er verschijnt een dialoogvenster waarin u wordt gevraagd de naam voor dit script te typen. Typ een naam en kies 'OK' om het script op te slaan op schijf.

5. Net als een formulier kunt u een script opslaan door 'Bestand | Opslaan' of 'Bestand | Opslaan als' te kiezen, en het script aanmaken door 'Taal | Aanmaken' te kiezen. Opgeslagen scripts kunnen worden veranderd; aangemaakte scripts niet. Raadpleeg Hoofdstuk 21 voor meer informatie over het aanmaken van formulieren en scripts.

# Script afspelen

U kunt een script afspelen door Paradox interactief te gebruiken of door het script aan te roepen vanuit een methode. In beide gevallen voert u de **run**-methode van het script uit.

## Paradox interactief gebruiken

Als u een script wilt starten door Paradox interactief te gebruiken, gaat u als volgt te werk:

1. Kies 'Bestand | Openen | Script'.
2. Er verschijnt een dialoogvenster met een lijst met beschikbare scripts. Kies een script uit de lijst, kies 'Afspelen'en kies 'OK'. Het script wordt uitgevoerd.

## Een script afspelen vanuit een methode

U gebruikt de **play**-methode van het System-type om een script af te spelen vanuit een methode of procedure. Bijvoorbeeld:

```
switch
   case deWaarde = "dit"  : play("vanAlles")        ; speel script "vanAlles" af
   case deWaarde = "dat"  : play("nogWat")          ; speel script "nogWat" af
   otherwise              : play("enEenBeetjeMeer") ; speel script
"enEenBeetjeMeer" af
endSwitch
```

HOOFDSTUK 21

# ObjectPAL-applicaties samenstellen en aanmaken

In dit hoofdstuk worden de concepten en procedures gepresenteerd die nodig zijn Paradox-applicaties gereed te maken voor eindgebruikers. Dit proces bestaat uit het verzamelen van alle bestanden die voor de applicatie nodig zijn, met inbegrip van formulieren, rapporten, tabellen, indexbestanden en eventueel queries. Verder komen de integratie van het Helpsysteem van Windows, de aanpassing van een applicatie voor internationale gebruikers en de documentatie van de applicatiecode aan de orde.

In dit hoofdstuk worden de volgende onderwerpen behandeld:

- Componenten van een Paradox-applicatie
- Formulieren opslaan en aanmaken
- Een Helpsysteem toevoegen
- Lokalisatie en tekensets
- De applicatie documenteren

## De applicatiecomponenten verzamelen

De hoofdcomponenten van een Paradox-applicatie zijn:

- Bureaublad
- Gegevensmodel-objecten
- Formulieren

## Bureaublad

Het bureaublad is het kader waarbinnen de Paradox-applicatie werkt. Het bureaublad zorgt voor een omgeving met het hoofdelement-venster, de menu's, de TurboBalk, een standaardsessie en lijsten met wachtwoorden. Kenmerken bepalen hoe een bureaublad eruitziet en hoe bepaalde aspecten van het bureaublad zich verhouden tot het systeem. Deze kenmerken zijn onder meer:

*Kenmerken bureaublad*

- Titelbalk van de applicatie
- Standaardfont
- Kleuren voor verschillende standaardcomponenten voor besturing en vensters
- Achtergrond
- Database
- Privé-directory
- Formaat van subelement-vensters
- Positie van de TurboBalk
- Liniaalstijl

Het bureaublad voor Paradox-ontwikkeling kan extra kenmerken hebben, zoals een start/ontwerpmodus en voorkeuren voor het opsporen van fouten en de bewerking.

***Opmerking*** Zie Hoofdstuk 14 in *Aan de slag* voor meer informatie over de kenmerken van het bureaublad. Zie ook het bestand SETTINGS.TXT in uw Paradox-systeemdirectory.

*Het bureaublad houdt wachtwoorden bij*

Het bureaublad houdt een lijst met wachtwoorden bij die door de gebruiker zijn ingevoerd en geannuleerd tijdens de huidige sessie.

Telkens als u Paradox aanroept, roept u het bureaublad opnieuw aan. Elk geactiveerd bureaublad van Paradox kan slechts één applicatie tegelijk starten. Paradox kan echter meerdere keren tegelijk op het zelfde Windows-platform gestart worden en op die manier meerdere tegelijk werkende Paradox-applicaties ondersteunen.

Elke keer dat u Paradox op hetzelfde werkstation start, start u een onafhankelijk programma, alsof u Paradox niet al eerder had gestart, dat wedijvert om het verkrijgen van resources en vergrendelingen, net als bij aparte werkstations.

***Opmerking*** Zie Hoofdstuk 17 voor meer informatie over de Session-variabele als u meerdere sessies wilt openen met ObjectPAL.

### Gegevensmodel-objecten

De gegevensmodel-objecten zijn Table en Query. Gegevensmodel-objecten worden niet automatisch opgenomen als u uw formulieren "aanmaakt". U moet deze dus expliciet opnemen als u uw produkt samenstelt. Elke tabel in het gegevensmodel wordt vertegenwoordigd door een of meer bestanden met bestandsnaamextensies die worden beschreven in Tabel 21-1.

Table 21-1 Extensies van bestanden die zijn verbonden met tabellen

| Functie | Paradox | dBASE |
|---|---|---|
| Tabel | .DB | .DBF |
| Primaire index | .PX | NVT |
| Secundaire index | .Xnn and .Ynn | .NDX or .MDX |
| Validiteitscontroles | .VAL | NVT |
| Memovelden | .MB | .DBT |

### Formulieren

In dit hoofdstuk betekent de term "formulier" "formulier, rapport, bibliotheek en script", tenzij anders wordt aangegeven. Formulieren worden in een "uitgeklede" versie aangemaakt via een procedure die "aanmaken" wordt genoemd. Deze procedure wordt in de volgende paragraaf besproken.

## Formulieren opslaan en aanmaken

Paradox kent twee manieren om formulieren te bewaren: opgeslagen en aangemaakt. Normaal gesproken *slaat* u formulieren *op* tijdens de ontwikkeling van applicaties en *maakt* u formulieren *aan* als onderdeel van de samenstelling van de applicatie voor aflevering en installatie bij eindgebruikers. Beide opties zijn alleen beschikbaar als u in een ontwerpvenster werkt.

U gebruikt de optie 'Opslaan' tijdens de ontwikkeling van de applicatie. De objecten en de code in opgeslagen formulieren kunnen alleen worden bewerkt met de ontwikkelingsversie van Paradox. Paradox kent bestandsnaamextensies toe aan opgeslagen formulierobjecten, zoals Tabel 21-2 laat zien.

U gebruikt de aanmaakoptie als uw formulier klaar is en u het kunt samenstellen voor eindgebruikers. Het aangemaakte formulier bevat geen broncode en de objecten en de code van het formulier kunnen niet worden bewerkt. Paradox kent bestandsnaamextensies toe aan aangemaakte formulieren, zoals Tabel 21-2 laat zien.

Als u een formulier opslaat of aanmaakt, wordt er een speciaal Windows-bestand, een DLL (Dynamic Link Library), gemaakt, dat

een of meer gecompileerde objecten en gecompileerde ObjectPAL-code bevat. Als u een formulier opslaat, wordt de ObjectPAL-broncode behouden; als u een formulier aanmaakt, gebeurt dat niet.

Table 21-2 Extensies van opgeslagen en aangemaakte objecten

| Object | Opgeslagen | Aangemaakt |
|---|---|---|
| Formulier | .FSL | .FDL |
| Bibliotheek | .LSL | .LDL |
| Rapport | .RSL | .RDL |
| Script | .SSL | .SDL |

*Prestaties*

U krijgt de beste prestaties als u de opties 'Debug-instructie mogelijk', 'Uitvoering volgen', 'Ingebouwde methodes volgen' en 'Ctrl-Break naar debugger' uitschakelt. Raadpleeg Hoofdstuk 10 voor meer informatie over deze opties.

## Formulieren opslaan

Als u een formulier, rapport, bibliotheek of script wilt opslaan, kiest u 'Bestand | Opslaan' (of 'Bestand | Opslaan als'). Geef het object een naam als hiernaar wordt gevraagd.

## Formulieren aanmaken

De stappen voor het aanmaken van formulieren lijken veel op die voor objecten. Er is echter een aantal verschillen. Deze stappen worden daarom afzonderlijk behandeld.

*Formulier*

Als u een formulier aanmaakt, gaat u als volgt te werk:

1. Open het formulier in een ontwerpvenster.
2. Kies 'Formulier | Aanmaken'.

*Rapport*

Als u een rapport aanmaakt, gaat u als volgt te werk:

1. Open het rapport in een ontwerpvenster.
2. Kies 'Rapport | Aanmaken'.

*Bibliotheek of script*

Als u een bibliotheek of een script aanmaakt, gaat u als volgt te werk:

1. Open de bibliotheek of het script in een ontwerpvenster.
2. Open een venster van de ObjectPAL-Editor.
3. Kies 'Taal | Aanmaken'.

## Helpsysteem toevoegen

ObjectPAL beschikt over een manier om de standaard Helpapplicatie van Windows aan te roepen. Dit is een zelfstandige Windows-applicatie. Als u Windows Help wilt gebruiken, maakt u eerst een bestand met Helpinformatie en contextreeksen, dat u vervolgens compileert met de Help-compiler van Microsoft (die wordt geleverd bij Borland C++). De documentatie die bij de Help-compiler wordt geleverd, beschrijft gedetailleerd hoe u een Windows Helpsysteem maakt en compileert.

Als u uw Helpsysteem hebt gecompileerd, kunt u de volgende methodes van het System-type gebruiken om toegang te krijgen tot het systeem:

- **helpOnHelp** geeft informatie over het gebruik van de Helpapplicatie.
- **helpQuit** vertelt de Helpapplicatie dat een bepaald Helpbestand niet wordt gebruikt.
- **helpSetIndex** vertelt de Helpapplicatie welke index moet worden gebruikt.
- **helpShowContext** toont de Helpinformatie die is geassocieerd met de opgegeven contextidentificatie.
- **helpShowIndex** toont de index van een bepaald Helpbestand.
- **helpShowTopic** toont de Helpinformatie die is geassocieerd met de opgegeven onderwerpsleutel.
- **helpShowTopicInKeywordTable** toont Helpinformatie die is geassocieerd met een sleutelwoord dat is opgeslagen in een alternatieve sleutelwoordtabel.

***Belangrijk*** In de MAST-applicatie en de online voorbeelden worden voorbeelden gegeven van hoe u toegang kunt krijgen tot een Helpsysteem van Windows. U kunt ook de informatie over deze methodes bekijken in de online ObjectPAL Help

## Lokalisatie voor internationale gebruikers

In deze paragraaf wordt besproken hoe u applicaties wijzigt voor internationale gebruikers.

## Internationale opmaak van numerieke constanten

De meeste ObjectPAL-uitdrukkingen gebruiken numerieke constanten in de Amerikaanse notatie.

Getallen en valutawaarden kunnen worden ingevoerd en weergegeven met behulp van een internationale getalopmaak. Deze opmaak kunt u opgeven met **formatSetCurrencyDefault** en andere opmaakprocedures van het System-type. Uw applicatie gebruikt standaard de internationale instelling van Windows.

***Opmerking*** De methodes **readProfileString** en **writeProfileString** van het System-type geven toegang tot het bestand WIN.INI van de gebruiker.

## Reeksen tussen aanhalingstekens

Reeksen tussen aanhalingstekens die met een formulier zijn geassocieerd, worden automatisch opgeslagen als resources. U hebt geen apart Windows-resource-bestand (.RC) nodig. Omdat het formulier een DLL is, kunt u de Resource Workshop van Borland gebruiken om deze reeksen te veranderen (bijvoorbeeld vertalen), zonder deze opnieuw te compileren of zonder dat u de broncode van de applicatie nodig hebt.

Stel dat u een formulier wilt aanmaken om de broncode te beveiligen, maar dat de berichten wel moeten kunnen worden vertaald. U kunt de berichten dan als reeksen tussen aanhalingstekens toewijzen aan String-constanten, bijvoorbeeld als volgt:

```
const
   ZoekBestand = "Zoek klant"
   NietGevonden = "Niet gevonden"
endConst
```

Vervolgens kunt u deze constanten gebruiken in instructies, bijvoorbeeld als volgt:

```
msgStop(ZoekBestand, NietGevonden + klantTC.Naam)
```

Als u het formulier hebt aangemaakt, kunnen de gebruikers de broncode niet wijzigen. Ze kunnen wel een resource-editor gebruiken om de tekst van de constanten 'ZoekBestand' en 'NietGevonden' te veranderen en indien nodig te vertalen.

U kunt dezelfde werkwijze gebruiken om naar velden in een tabel te verwijzen. Het volgende voorbeeld laat zien dat als u een veldnaam tussen aanhalingstekens plaatst, deze veldnaam een resource wordt. Stel dat u code schrijft die de waarde van het veld 'Straat' uit de tabel *Klant* gebruikt. Als deze applicatie in Frankrijk zou worden gebruikt, zouden alle veldnamen naar het Frans worden vertaald. Het veld 'Straat' zou de naam 'Rue' krijgen. Als uw code reeksen tussen aanhalingstekens als veldnamen heeft gebruikt, kunnen de verwijzingen met behulp van een resource-editor worden vertaald. De applicatie blijft dan werken.

```
proc eenProc
Var tc TCursor endVar
   tc.open("Klant.db")
   x = tc."Straat"   ; "Straat" wordt resource en kan worden bewerkt
   y = tc.Straat     ; Straat wordt geen resource en kan niet worden bewerkt
endProc
```

## Formulieren

Etiketten en tekst op het formulier zelf worden geen resource. Deze worden dus niet als resources opgeslagen en kunnen niet worden bewerkt met applicaties zoals de Resource Workshop van Borland. Hierdoor kan het moeilijk zijn om uw formulieren (en de tekst op de formulieren) naar een andere taal te vertalen.

U kunt dit probleem met ObjectPAL oplossen door de tekst van etiketten en tekstobjecten tijdens de initialisatieprocedure in te stellen in plaats van harde code te schrijven voor de tekst van het object. U plaatst de code in de **open**-methode van alle objecten en gebruikt reeksen tussen dubbele aanhalingstekens, zodat de reeksen resources worden, zoals in de vorige paragraaf.

## Tekensets

Paradox herkent twee soorten tekensets: OEM en ANSI. Een OEM-set (ook een *codepagina* geheten) bevat 256 tekens die zijn genummerd van 0 tot 255. De tekens 0 tot en met 127 zijn in elke codepagina hetzelfde. De tekens 128 tot en met 255 worden uitgebreide tekens genoemd en zijn voor elke codepagina anders. De uitgebreide tekens bevatten accenttekens en speciale symbolen voor verschillende talen. DOS 5.0 bevat verschillende codepagina's, waarvan er slechts één tegelijk kan worden geactiveerd. De codepagina's in DOS 5.0 zijn: Verenigde Staten, Meertalig (Latijns I), Slavisch (Latijns II), Portugees, Frans-Canadees en Scandinavisch.

Paradox herkent één set met ANSI-tekens. De tekens 32 tot en met 126 komen overeen met de tekens van de OEM-tekensets. Andere tekens kunnen zich op andere plaatsen in OEM-codepagina's bevinden of volledig ontbreken. De code voor het teken ñ is bijvoorbeeld 164 in de Engelse OEM-set, maar 241 in de ANSI-set.

Paradox en dBASE slaan tabellen op met de OEM-tekenset, terwijl Windows (en dus ook Paradox zelf) ANSI-tekens gebruikt. Paradox gebruikt het taalaansturingsbestand (zie het *Handboek*) voor sorteringen en andere tekenbewerkingen. De taalaansturing heeft geen invloed op de datum-, getal- en valutaopmaak.

Er kunnen problemen ontstaan omdat OEM-codepagina's en de ANSI-tekenset niet dezelfde tekens bevatten. Paradox zorgt voor de OEM–ANSI-conversie als uit tabellen wordt gelezen of naar tabellen wordt geschreven, maar in andere gevallen (als bijvoorbeeld naar een tekstbestand wordt geschreven of gegevens worden uitgewisseld via een DDE-koppeling) moet u, de programmeur, zelf de procedure beheren. Tabel 21-3 geeft een overzicht van de ObjectPAL-methodes

die u voor dit doel kunt gebruiken. De weergegeven methodes behoren alle tot het String-type.

Table 21-3 Methodes voor OEM-ANSI conversie

| Methode | Beschrijving |
|---|---|
| ansiCode | Geeft de geheel-getalwaarde van een teken uit de ANSI-tabel terug |
| chr | Converteert een ANSI-code naar een ANSI-teken |
| chrOEM | Converteert een ANSI-code naar een OEM-reeks van één teken |
| oemCode | Geeft de geheel-getalwaarde van een teken uit de OEM-tabel terug |
| toANSI | Converteert een OEM-tekenreeks naar ANSI-tekens |
| toOEM | Converteert een ANSI-tekenreeks naar OEM-tekens |

# De applicatiecode documenteren

ObjectPAL voorziet in verscheidene methodes en procedures om de objecten en methodes bij te houden die zich op een formulier bevinden. Deze methodes en procedures zijn gedefinieerd voor het UIObject-type en het Form-type en beginnen allemaal met het voorvoegsel "enum", een afkorting van "enumerate" (opsommen), zoals in **enumSource**.

De **enumSource**-methode maakt een Paradox-tabel die de namen bevat van alle objecten waaraan methodes zijn gekoppeld, en de broncode van elke methode. De **enumSourceToFile**-methode doet hetzelfde, maar schrijft de gegevens naar een tekstbestand in plaats van naar een tabel.

Als u een tabel wilt maken met een overzicht van alle objecten op een formulier, gebruikt u **enumUIObjectNames**, ongeacht of de methodes zijn gekoppeld.

*Interactieve functie*

U kunt Paradox ook interactief gebruiken om een rapport te maken met de broncode en de objecten waaraan de code is gekoppeld. De ObjectPAL-Editor kent de functie 'Bronnen doorbladeren', die een snel rapport maakt van de broncode op een formulier. Als u deze functie wilt gebruiken, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Bronnen doorbladeren'. Zie Hoofdstuk 9 voor meer informatie over de ObjectPAL-Editor.

DEEL V

# Appendixen

Dit deel van de handleiding bevat referentiemateriaal voor ObjectPAL-programmeurs.

- In Appendix A, "PAL en ObjectPAL", worden een aantal verschillen tussen PAL en ObjectPAL besproken.
- In Appendix B, "Ingebouwde methodes", worden de ingebouwde methodes van ObjectPAL besproken. Ingebouwde methodes bepalen hoe objecten reageren op acties.
- In Appendix C, "Kenmerken", worden de kenmerken van alle UIObjecten opgesomd, en worden alle waarden vermeld die elk kenmerk kan aannemen.

APPENDIX A

# PAL en ObjectPAL

Deze appendix geeft een overzicht van de verschillen tussen ObjectPAL en PAL, met als doel PAL-programmeurs te wijzen op belangrijke verschillen in inhoud en invalshoeken tussen deze twee programmeeromgevingen. Raadpleeg ook het *Handboek* en *Aan de slag*.

In deze appendix worden de volgende onderwerpen behandeld:

- Programmeeromgeving
- Taalverschillen
- Tabellen

## Programmeeromgeving

De overschakeling van PAL-programmeren op ObjectPAL-programmeren vereist een verandering in de manier waarop u aankijkt tegen programmeren. Deze paragraaf geeft een overzicht van de belangrijkste verschillen tussen de programmeeromgevingen.

### Proceduregericht versus objectgeoriënteerd

PAL is een proceduregerichte taal. Een PAL-applicatie kan dus worden beschreven als een lineaire opeenvolging van opdrachten en functies. ObjectPAL daarentegen is objectgeoriënteerd en ObjectPAL-applicaties zijn actiegestuurd. Voor programmeurs die zijn gewend aan procedurele talen vraagt ObjectPAL om een andere invalshoek bij het ontwerpen van een programma en het schrijven van de code. Zie Hoofdstuk 8 voor een overzicht van objecten en acties en van de objectgeoriënteerde programmeerstrategie.

### Programmering gestuurd door gebruikersinterface

In Paradox voor DOS werkt een PAL-programma als een geautomatiseerde gebruiker. Dat wil zeggen, het PAL-programma bootst een gebruiker na van Paradox voor DOS en gebruikt de applicatie in feite door de interface te besturen als een

geautomatiseerde gebruiker achter de schermen. Het effect van PAL-opties op objecten in het werkgebied blijft normaal gesproken verborgen voor de gebruiker, tenzij u de opdracht ECHO geeft.

Als u echter een ObjectPAL-applicatie maakt, *maakt* u feitelijk de gebruikersinterface en definieert u het gedrag van de interface. ObjectPAL biedt nieuwe manieren om tabellen te manipuleren zonder deze op het scherm van de gebruiker weer te geven. ObjectPAL kent dus geen concept dat overeenkomt met de PAL-opdracht ECHO.

## Actiemodel

Het actie- en handelingmodel van ObjectPAL werkt heel anders dan de actie- en trigger-mechanieken van PAL (PAL 4.0). U dient te weten hoe het actiemodel werkt om te begrijpen hoe ObjectPAL-applicaties werken. Zie Hoofdstuk 10 voor gedetailleerdere informatie over de manier waarop ObjectPAL acties behandelt.

## Scripts en formulieren

Als u een PAL-applicatie maakt, ligt het accent op het *script*. De applicatie bestaat uit een of meer scripts met PAL-instructies die achtereenvolgens worden uitgevoerd en die het gedrag van het programma bepalen.

Als u een ObjectPAL-applicatie maakt, ligt het accent op het *formulier*. De applicatie bestaat uit objecten die op het formulier zijn geplaatst. U kunt het standaardgedrag van deze objecten veranderen door ObjectPAL-code te koppelen aan deze objecten en aan het formulier zelf. Het gecombineerde gedrag van het formulier en de objecten op het formulier definieert vervolgens het gedrag van het programma.

PAL en ObjectPAL gebruiken de termen *formulier* en *script* om verschillende concepten te beschrijven. PAL plaatst applicatiecode in het script en gebruikt formulieren om tabellen weer te geven en voor de referentiële integriteit te zorgen. ObjectPAL koppelt applicatiecode aan objecten op het formulier en gebruikt het scriptobject als een plaats voor ObjectPAL-code die niet is gekoppeld aan formulieren. Met PAL kunnen scripts worden opgenomen, maar ObjectPAL-scripts worden altijd geschreven. In Hoofdstuk 13 en Hoofdstuk 14 worden formulieren behandeld en in Hoofdstuk 20 worden scripts behandeld.

## Dialoogvensters

In Paradox voor Windows is een dialoogvenster een speciaal type formulier. Deze dialoogvensters hebben andere eigenschappen en mogelijkheden dan de dialoogvensters van Paradox voor DOS. In hoofdstuk 14 worden formulieren en dialoogvensters behandeld.

## Bibliotheken

Bibliotheken werken in ObjectPAL volgens hetzelfde principe als de procedurebibliotheken van PAL. Beide soorten bibliotheken besparen geheugen en vergemakkelijken het onderhoud van eigen code. ObjectPAL-bibliotheken kunnen methodes, procedures (lokaal voor

de bibliotheek), constanten en gedeclareerde variabelen bevatten. In ObjectPAL wordt het geheugenbeheer automatisch verzorgd door de Windows-omgeving.

### Bureaublad, werkgebied, canvas

In PAL wordt het *canvas* gebruikt om uitvoer te tonen aan de gebruiker. De termen *canvas* en *werkgebied* worden niet gebruikt in Paradox voor Windows en de term 'bureaublad' heeft een andere betekenis. In hoofdstuk 14 worden de ObjectPAL-methodes behandeld voor het werken met het bureaublad.

### Hiërarchie van ingesloten objecten

De objecten op een Paradox voor Windows-formulier vormen een *hiërarchie van ingesloten objecten*. De hiërarchie van ingesloten objecten is gebaseerd op de visuele ruimtelijke relaties tussen objecten op het formulier en bepaalt het bereik van variabelen en eigen code. Zie hoofdstuk 13 voor meer informatie over insluiting.

## Taal

In deze paragraaf wordt een aantal belangrijke verschillen tussen de PAL-taal en de ObjectPAL-taal behandeld.

### Taalelementen

PAL-scripts bestaan uit opdrachten, functies, sleutelwoorden en eigen procedures. In PAL worden ingebouwde mogelijkheden aangeboden via opdrachten en functies en wordt de term 'procedure' alleen gebruikt voor functies die zijn gedefinieerd door de gebruiker. De opdrachten omvatten controlestructuren, toets- en menu-equivalenten en opdrachten voor de manipulatie van gegevens, geheugen en invoer en uitvoer. Functies verschillen syntactisch gezien van opdrachten en een functie geeft altijd een waarde terug. Functies voeren wiskundige conversies, statistische conversies of gegevenstype-conversies uit of geven statusinformatie terug. De programmeur moet er zelf voor zorgen dat een functie in een juiste context wordt gebruikt.

De ObjectPAL-taal bestaat uit ingebouwde methodes, RTL(runtime library)-methodes en -procedures, elementaire taalonderdelen en eigen methodes en procedures. Methodes en procedures geven gewoonlijk een waarde of een indicatie van succes of mislukking terug. Eigen methodes en procedures komen overeen met door de gebruiker gedefinieerde functies in PAL.

ObjectPAL-methodes worden ingedeeld volgens het objecttype waarop de methodes werken (Form, Table, UIObject enzovoort). Het object waarop een methode werkt, maakt deel uit van de syntaxis van

de methode-aanroep. Procedures werken op dezelfde manier, maar bevatten geen objectnaam in de aanroep.

Elk object op een Paradox-formulier heeft een eigen verzameling ingebouwde methodes. De ObjectPAL-programmeur kan deze met eigen code overschrijven, wat een gebruikelijke manier is om standaardgedrag te veranderen. In Hoofdstuk 11 wordt de taalstructuur van ObjectPAL behandeld.

## Bereik van variabelen

PAL-variabelen hebben een globaal bereik, met uitzondering van variabelen die formele parameters zijn, variabelen die expliciet worden gedeclareerd als *lokaal* en variabelen die binnen gesloten procedures worden gedeclareerd. Daarnaast gebruiken PAL-programmeurs het concept *dynamisch bepalen van het bereik.*

In ObjectPAL gelden voor het bereik van variabelen bepaalde beperkingen, die in Hoofdstuk 11 worden besproken. Het bereik van variabelen wordt bepaald door de hiërarchie van ingesloten objecten.

## Variabelen declareren

In PAL worden variabelen niet gedeclareerd. In ObjectPAL verbetert de declaratie van variabelen de prestaties en is het vaak een vereiste.

## Constanten en kenmerken

De ObjectPAL-IDE verschaft toegang tot honderden door het systeem gedefinieerde constanten en kenmerken, die het programmeren vergemakkelijken en ervoor zorgen dat programma's gemakkelijker te lezen en te onderhouden zijn. Veel PAL-functies zijn doeltreffend vervangen door ObjectPAL-kenmerken.

## Gebruikersinvoer

De *accept*-instructie speelt een belangrijke rol in het krijgen van invoer van de gebruiker in PAL-programmering.

In ObjectPAL krijgt de programmeur gebruikersinvoer door middel van veldobjecten, door eigen formulieren en dialoogvensters te maken of door een van de dialoogvensterprocedures van het System-type aan te roepen, zoals **msgInfo**, **message** en **msgYesNoCancel**. Daarnaast accepteren bepaalde **view**-methodes gebruikersinvoer.

## Codebeheer

PAL wordt geïnterpreteerd, maar niet gecompileerd. PAL-procedures kunnen in bibliotheken worden opgeslagen om de prestaties te verbeteren. PAL-broncode wordt opgeslagen in scriptbestanden.

ObjectPAL-code wordt gecompileerd en opgeslagen in Windows-DLL's. Als u een ObjectPAL-applicatie aanmaakt, wordt de broncode uit de DLL verwijderd.

### Foutverwerking

In PAL wordt de systeemvariabele *errorproc* ingesteld op de naam van de foutbehandelingsprocedure die de gebruiker heeft gedefinieerd.

ObjectPAL daarentegen biedt verschillende mogelijkheden voor de behandeling van fouten: kritieke fouten, waarschuwingsfouten, de ingebouwde **error**-methode en het TRY...FAIL-blok. ObjectPAL kent ook een *foutstapel* en verscheidene met deze stapel verbonden RTL-foutmethodes.

### Andere systeemvariabelen

Twee andere belangrijke PAL-systeemvariabelen, *retval* en *autolib*, worden niet gebruikt in ObjectPAL.

ObjectPAL kent een logisch gegevenstype en bibliotheekobjecten.

### Standaardopmaken

PAL kent een aantal standaardopmaken die afwijken van de opmaken in ObjectPAL. Paradox voor Windows gebruikt de standaarden van Windows. De programmeur kan in beide produkten de standaarden vervangen.

### copyToArray en copyFromArray

In PAL kopiëren de opties COPYFROMARRAY en COPYTOARRAY de tabelnaam als het eerste element van de array. Daarna worden de veldgegevens als volgende elementen gekopieerd.

De ObjectPAL-methodes **copyFromArray** en **copyToArray** kopiëren alleen veldgegevens. De tabelnaam wordt verstrekt door de objectvariabele (TCursor of UIObject).

### Query-variabelen

In PAL wordt een lege tilde-variabele behandeld als een lege waarde. ObjectPAL behandelt een lege tilde-variabele als een null-waarde.

### Datumberekeningen

Als u in PAL een datum aftrekt van een andere datum, krijgt u als resultaat een getalwaarde (Date - Date = Number). In ObjectPAL krijgt u een datum (Date - Date = Date).

## Tabellen

Deze paragraaf geeft een overzicht van enkele belangrijke verschillen die vooral te maken hebben met gegevens in tabellen.

### Toegang tot en manipulatie van gegevens

In PAL moet u tabellen weergeven in werkgebied-objecten (als ECHO is uitgeschakeld) om toegang te krijgen tot de tabellen en deze te manipuleren.

In ObjectPAL krijgt u daarentegen directe programmeertoegang tot tabellen. ObjectPAL kent de variabeletypes TCursor en Table, waardoor u achter de schermen toegang kunt krijgen en tabellen kunt manipuleren. In hoofdstuk 16 wordt uitgelegd hoe u met gegevens in tabellen werkt.

## Referentiële integriteit

Paradox voor DOS gebruikt het *formulier* voor multi-tabel referentiële integriteit.

Paradox voor Windows past regels voor gegevensintegriteit over meerdere tabellen toe, ongeacht de constructie van het formulier. Referentiële integriteit wordt hier beheerd door de interactieve Table-hulpmiddelen of door expliciete programmeerinstructies van ObjectPAL (zie de beschrijving van **create** in de online ObjectPAL Help).

## Samen bewerken

De bewerkmodus van ObjectPAL komt overeen met de modus *Samen bewerken* van PAL.

## Tabelverwanten

Paradox voor Windows kent het formele concept tabelverwanten van Paradox voor DOS niet. Het belangrijkste verschil is dat formulieren van Paradox voor Windows *niet per se gekoppeld* hoeven te zijn aan een of meer tabellen. Tabellen kunnen tijdens de uitvoering worden verbonden en formulieren kunnen bestaan zonder enige tabelverbinding.

Als u een Paradox voor Windows-applicatie wilt verzenden, controleer dan of u de noodzakelijke groepen bestanden hebt opgenomen. (Raadpleeg Hoofdstuk 21 voor meer informatie.) De IDE bevat tabelhulpmiddelen, bijvoorbeeld voor kopiëren en verwijderen.

## Inkorting veldbreedte

Als u gegevens probeert in te voegen in een veldobject, en de velddefinitie korter is dan de gegevens, kort PAL de gegevens in, waarna de uitvoering wordt vervolgd. In ObjectPAL veroorzaakt deze poging een runtime fout.

APPENDIX B

# Ingebouwde methodes

In deze appendix worden de ingebouwde methodes voor Paradox-objecten beschreven.

In deze appendix komen de volgende onderwerpen aan de orde:

- Over ingebouwde methodes: beschrijft het koppelen van code aan een ingebouwde methode en de controle over het standaard-gedrag van een object.
- Beschrijvingen van de ingebouwde methodes: beschrijft de ingebouwde methodes voor interne en externe acties.
- Speciale ingebouwde methodes: beschrijft ingebouwde methodes die uniek zijn voor bepaalde objecten.
- Ingebouwde objectvariabelen: beschrijft ObjectPAL-variabelen voor het verwijzen naar objecten op een formulier.

***Belangrijk*** Voordat u deze appendix leest, moet u vertrouwd zijn met het actie-model van ObjectPAL, dat wordt beschreven in Hoofdstuk 12.

## Over ingebouwde methodes

Elk object op een formulier (en het formulier zelf) heeft ingebouwde methodes voor de behandeling van acties. Ingebouwde methodes hebben dezelfde naam als de acties die deze methodes activeren. Als bijvoorbeeld een waarde wordt veranderd, wordt de ingebouwde **changeValue**-methode van een object geactiveerd; als de muisknop wordt ingedrukt, wordt de ingebouwde **mouseDown**-methode geactiveerd en als de muisknop wordt losgelaten, wordt de ingebouwde **mouseUp**-methode geactiveerd. Het gedrag van een object bestaat uit de combinatie van de ingebouwde methodes van het object.

ObjectPAL kent drie soorten ingebouwde methodes:

- Ingebouwde methodes voor interne acties—acties die intern door ObjectPAL of Paradox worden gegenereerd
- Ingebouwde methodes voor externe acties—acties die meestal worden gegenereerd door handelingen van de gebruiker (deze acties kunnen echter ook worden gegenereerd door ObjectPAL-instructies)
- Speciale ingebouwde methodes—methodes die zijn ingebouwd in enkele specifieke objecten

In deze paragraaf worden ingebouwde methodes en ingebouwde variabelen besproken. Daarna worden enkele eenvoudige voorbeelden gegeven van het gebruik van deze methodes tijdens de bewerking van gegevens.

### Code koppelen aan ingebouwde methodes

U koppelt code aan een ingebouwde methode door een venster van de ObjectPAL-Editor te openen en de code te typen. Elk ontwerpobject heeft bijvoorbeeld een ingebouwde **mouseClick**-methode die bepaald standaardgedrag uitvoert als u op dat object klikt (zie verderop). Als u dat gedrag wilt veranderen, inspecteert u het object en kiest u 'Methodes' in het kenmerkmenu. Vervolgens kiest u in de lijst met methodes **mouseClick** om een venster van de ObjectPAL-Editor te openen. Typ uw code en sla deze op. Uw code wordt nu uitgevoerd als de **mouseClick**-methode van dit object wordt aangeroepen.

De ingebouwde code wordt ook uitgevoerd, echter *na* uw code (vlak voor het sleutelwoord **endmethod**).

De ingebouwde code is impliciet en wordt automatisch uitgevoerd. U kunt het standaardgedrag echter veranderen, bijvoorbeeld als u de ingebouwde code wilt aanroepen *vóór* uw code of als u wilt voorkomen dat de ingebouwde code wordt uitgevoerd (zie verderop in dit hoofdstuk). Eerst moet u echter het standaardgedrag van de ingebouwde methodes begrijpen.

## Beschrijvingen van de ingebouwde methodes

Tabel B-1 geeft een overzicht van de ingebouwde methodes voor interne en externe acties en van de speciale ingebouwde methodes. Achter de tabel volgen beschrijvingen waarin wordt aangegeven wanneer de methodes worden aangeroepen, wat het standaardgedrag is en wat de (eventuele) effecten van een fout zijn.

Deze paragraaf is gebaseerd op materiaal dat is gepresenteerd in Hoofdstuk 12 en Hoofdstuk 13.

Tabel B-1 Ingebouwde methodes

| Intern | Extern | Speciaal |
|---|---|---|
| open | mouseClick | pushButton |
| close | mouseDown | changeValue |
| canArrive | mouseUp | newValue |
| arrive | mouseDouble | |
| setFocus | mouseRightDown | |
| canDepart | mouseRightUp | |
| removeFocus | mouseRightDouble | |
| depart | mouseMove | |
| mouseEnter | keyPhysical | |
| mouseExit | keyChar | |
| timer | error | |
| | status | |
| | action | |
| | menuAction | |

## Ingebouwde methodes voor interne acties

De volgende methodes worden geactiveerd door interne acties—acties die intern worden gegenereerd door ObjectPAL. Interne acties gaan, zoals alle acties, eerst naar het formulier. Het formulier stuurt de acties door naar het doelobject. Interne acties borrelen niet omhoog in de hiërarchie van ingesloten objecten.

*open*

**open** wordt eenmaal aangeroepen voor elk object op het formulier, te beginnen bij het formulier zelf, gevolgd door de ingesloten objecten. De **open**-methode van het formulier opent standaard alle tabellen die aan het formulier zijn gekoppeld.

Als bij een van de objecten een fout optreedt, kan het formulier niet worden geopend.

De standaardcode van de **open**-methode van een object roept de **open**-methode van alle subobjecten aan (de objecten die zich één niveau lager bevinden in de hiërarchie van ingesloten objecten). Met andere woorden, de **open**-methode van het formulier roept standaard de **open**-methode aan van alle pagina's op het formulier en de **open**-methode van de pagina's roepen de **open**-methode aan van alle objecten op de pagina, enzovoort.

*close*

**close** wordt eenmaal aangeroepen voor alle objecten op een formulier dat wordt gesloten. De **close**-methode van een formulier sluit standaard alle tabellen die aan het formulier zijn gekoppeld.

*canArrive*

**canArrive** wordt aangeroepen als de focus naar een object wordt verplaatst . Deze methode vraagt of het object (meestal een veldobject) actief mag worden. Paradox roept deze methode aan door

de insluitende objecten van het object af te gaan. Hierbij wordt **canArrive** voor elk object geactiveerd, totdat het object zelf wordt bereikt. Als op enig niveau in de hiërarchie een fout optreedt, wordt de toestemming geweigerd en wordt de verplaatsing ongedaan gemaakt.

Paradox blokkeert **canArrive** standaard voor records buiten de grenzen van een tabel (behalve voor lege records in de bewerkmodus). Dit geldt ook voor veldobjecten. Deze activeren ook het kenmerk Tab Stop, waardoor verplaatsing naar veldobjecten die geen tabstops zijn, wordt geblokkeerd. **canArrive** wordt ook geblokkeerd voor een veld als de gebruiker geen lees- en toegangsrechten heeft. Een kruistabulatie-object blokkeert **canArrive** als het doel geen cel is.

*arrive* **arrive** wordt alleen aangeroepen als het doel en de insluitende objecten van het doel een **canArrive** toestaan. Paradox roept **arrive**, net als **canArrive**, aan voor de insluitende objecten van het doelobject en ten slotte voor het doelobject zelf. Bij pagina's, tabelframes en multi-record objecten wordt de focus altijd naar het eerste tabstop-object verplaatst dat deze insluiten.

Een geslaagde **arrive** naar een record of een veldobject maakt van dit record of veldobject het huidige object (en opent, indien van toepassing, een bewerkvenster voor het veldobject).

**arrive** kan meer effecten op een veldobject hebben, afhankelijk van het weergavetype. Bij een afrollijst wordt de focus naar de lijst verplaatst. Bij een keuzeknop wordt de focus verplaatst naar de eerste knop.

*setFocus* **setFocus** wordt aangeroepen na een geslaagde **arrive** of wanneer de focus aan het formulier wordt teruggegeven na het aanroepen en weer verlaten van een ander venster. Deze methode wordt aangeroepen voor alle insluitende objecten van het actieve object, te beginnen met het buitenste insluitende object. Pas daarna wordt de methode aangeroepen voor het actieve object zelf.

In een bewerkveld zorgt de standaardcode van **setFocus** ervoor dat de huidige selectie wordt gemarkeerd en dat de invoegpositie begint te knipperen. Op dat moment wordt het kenmerk Focus van het object ingesteld op True en geeft het formulier een statusbericht weer met het nummer van het huidige record en het totale aantal records.

Als een knop een **setFocus** accepteert, verschijnt er een rechthoek om het label.

*canDepart* **canDepart** wordt aangeroepen bij een poging tot verplaatsing vanuit een object. Elke fout blokkeert de verplaatsing. Veldobjecten proberen de veldinhoud door te voeren (waardoor **changeValue** wordt geactiveerd) en recordobjecten proberen het huidige record door te

voeren als er veranderingen zijn aangebracht. Als het record is vergrendeld, roept het formulier **action(DataUnlockRecord)** aan.

*removeFocus*

**removeFocus** verwijdert de knipperende invoegpositie en de markering (indien van toepassing) van een veldobject en verwijdert de rechthoek om een knop. Het kenmerk Focus van het object wordt ingesteld op False.

Deze methode wordt voor het actieve object en voor alle insluitende objecten van het object (te beginnen met het actieve object) aangeroepen als de gebruiker een ander venster activeert of naar een ander object gaat.

*depart*

**depart** wordt aangeroepen nadat alle insluitende objecten van het huidige object via **canDepart** en **removeFocus** toestemming hebben gegeven het veldobject te verlaten.

Veldobjecten sluiten de bewerkvensters, werken het scherm bij en voeren algemene huishoudelijke taken uit.

*mouseEnter*

**mouseEnter** wordt aangeroepen als de aanwijzer binnen een object komt. Deze methode wordt alleen aangeroepen bij de overgang naar het object en niet bij elke verplaatsing over het object.

Veldobjecten veranderen de aanwijzer standaard in een I-vorm. Formulier-, pagina- en knopobjecten veranderen de aanwijzer standaard in een pijl.

Als een knop het laatste object is dat een klik heeft ontvangen en de muisknop nog ingedrukt is, schakelt de waarde van de knop tussen True en False.

*mouseExit*

**mouseExit** wordt aangeroepen als de aanwijzer een object verlaat. Een object dat de vorm van de aanwijzer bij **mouseEnter** instelt, verandert deze bij **mouseExit** in een pijl.

Als een knop het laatste object is dat een klik heeft ontvangen en de muisknop nog ingedrukt is, schakelt de waarde van de knop tussen True en False.

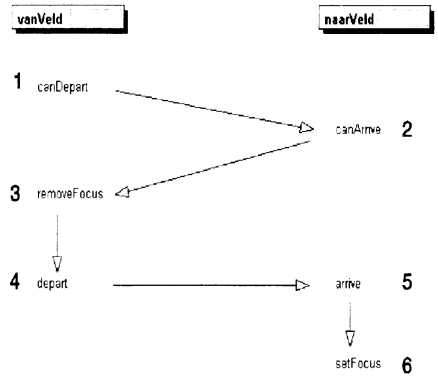
*timer*

**timer** wordt steeds aangeroepen als een timer-interval verstrijkt. Gebruik de UIObject-methode **setTimer** om timer-intervallen in te stellen.

## Volgorde van uitvoering

Afbeelding B-1 toont de volgorde waarin ingebouwde methodes worden uitgevoerd als u van het ene veldobject naar het andere gaat (bijvoorbeeld door op *Tab* te drukken). In dit voorbeeld heet het veldobject dat u verlaat, *vanVeld* en het veld waar u naartoe gaat, *naarVeld*.

Afbeelding B-1 Verplaatsen tussen veldobjecten




In deze afbeelding wordt de volgorde getoond waarin ingebouwde methodes worden uitgevoerd als de focus tussen twee veldobjecten wordt verplaatst.

## Ingebouwde methodes voor externe acties

De volgende methodes worden geactiveerd door externe acties. Deze acties worden meestal gegenereerd door handelingen van de gebruiker, maar kunnen ook worden gegenereerd door ObjectPAL-instructies. De verwerking van alle externe acties begint met het formulier, dat als verzendpunt fungeert. Als een externe actie niet wordt verwerkt door een object, borrelt deze omhoog in de hiërarchie van ingesloten objecten.

***Belangrijk*** Voor de meeste objecten en de meeste ingebouwde methodes houdt het standaardgedrag in dat de actie wordt doorgegeven in de hiërarchie van ingesloten objecten. Als een object iets anders doet, wordt dat in de volgende beschrijvingen aangegeven.

*mouseDown* **mouseDown** wordt aangeroepen als de linker muisknop wordt ingedrukt. Het actiepakket van deze methode bevat de muis-coördinaten in twips, in verhouding tot het laatste object waarvan de **mouseEnter**-methode is aangeroepen.

Een actief veldobject activeert de veldweergave, plaatst de invoegpositie en begint een sleepselectie.

Als het formulier een **mouseDown** verwerkt, wordt **mouseExit** aangeroepen voor alle objecten die zich niet meer onder de muis bevinden, en wordt **mouseEnter** aangeroepen voor alle objecten die zich nu wel onder de muis bevinden. Het formulier verstuurt de **mouseDown** naar het object waarnaar de muis wees.

Deze methode laat de waarde van een knop schakelen tussen True en False.

*mouseUp*

**mouseUp** wordt aangeroepen als de linker muisknop wordt losgelaten. Deze methode wordt aangeroepen voor het laatste object dat een **mouseDown** heeft ontvangen, zelfs als de knop buiten het object wordt losgelaten. Het object ziet dus altijd de combinatie **mouseDown/mouseUp**.

Een actief veldobject beëindigt de selectie; een veldobject dat niet actief is, voert een **self.moveTo()** uit.

Als het formulier een **mouseUp** verwerkt, wordt **mouseExit** aangeroepen voor alle objecten die zich niet meer onder de muis bevinden, en wordt **mouseEnter** aangeroepen voor alle objecten die zich nu wel onder de muis bevinden. Het formulier verstuurt de **mouseUp** vervolgens naar het object dat de laatste klik heeft ontvangen.

Deze methode laat de waarde van een knop schakelen tussen True en False. Als **mouseUp** wordt aangeroepen en de aanwijzer zich binnen een knop bevindt, wordt de **pushButton**-methode van de knop geactiveerd.

*mouseDouble*

**mouseDouble** wordt aangeroepen als met de linker muisknop wordt gedubbelklikt. Geheel volgens de Windows-conventie worden eerst een **mouseDown** en een **mouseUp** gegenereerd.

Veldobjecten activeren de veldweergave bij een **mouseDouble**.

Als het formulier een **mouseUp** verwerkt, wordt **mouseExit** aangeroepen voor alle objecten die zich niet meer onder de muis bevinden, en wordt **mouseEnter** aangeroepen voor alle objecten die zich nu wel onder de muis bevinden. Het formulier verstuurt de **mouseDouble** vervolgens naar het object dat de laatste klik heeft ontvangen.

*mouseRightDown*
*mouseRightUp*
*mouseRightDouble*

Deze methodes zijn equivalenten van de drie bovenstaande, maar hebben betrekking op de rechter muisknop.

De volgende veldobjecten geven bovendien een pop-up menu weer bij de ontvangst van een **rightMouseUp**: opgemaakt memo, afbeelding, OLE en niet-verbonden (ongedefinieerd).

*mouseClick*

**mouseClick** wordt aangeroepen als de logische linker muisknop wordt ingedrukt en losgelaten terwijl de aanwijzer zich binnen de grenzen van een object bevindt. **mouseClick** wordt niet aangeroepen als de gebruiker de muis buiten het object plaatst alvorens de muisknop los te laten.

**mouseClick** wordt gegenereerd vanuit de **mouseUp**-methode.

De vertaling van **mouseUp** naar **mouseClick** treedt op bij het eerste insluitende object dat iets met **mouseUp** doet. Met andere woorden, een **mouseUp** in een kader borrelt omhoog naar het insluitende object enzovoort. Alleen het veldobject, het knopobject, het lijstobject en het

formulier onderscheppen **mouseUp**. Dit zijn dus de plaatsen waar de vertaling plaatsvindt. Als u klikt op een object binnen een knop, wordt de **mouseClick** van dat object aangeroepen. Als dat object de standaard (borrelen) toestaat, ontvangt de knop uiteindelijk de **mouseClick**, waardoor de **pushButton** van de knop wordt geactiveerd. Zo kunt u code laten uitvoeren op objecten waarop u klikt binnen een knop, maar toch de **pushButton**-methode van de knop activeren. Als in **mouseUp** de foutcode wordt ingeschakeld, treedt **mouseClick** niet op en als in **mouseClick** de foutcode wordt ingeschakeld, treedt **pushButton** niet op.

*mouseMove*

**mouseMove** wordt aangeroepen als de muis binnen een object wordt verplaatst. Het actiepakket voor deze methode bevat de coördinaten van de aanwijzer (in twips).

Een actief bewerkveld kijkt naar de status van de *Shift*-toets. Als *Shift* (fysiek of logisch) wordt ingedrukt, wordt de selectie uitgebreid. In een actief grafisch veld schuift de afbeelding indien nodig.

Als u de muisknop binnen een object indrukt en ingedrukt houdt, wordt **mouseMove** aangeroepen totdat u de knop loslaat. Dit gebeurt ook als u de aanwijzer buiten het object plaatst.

Als het formulier een **mouseMove** verwerkt, wordt **mouseExit** aangeroepen voor alle objecten die zich niet meer onder de muis bevinden, en wordt **mouseEnter** aangeroepen voor alle objecten die zich nu wel onder de muis bevinden.

*keyPhysical*

**keyPhysical** wordt aangeroepen als een toets wordt ingedrukt en als een toets repeteert. De toetsaanslag gaat eerst naar het formulier en het formulier stuurt deze door naar het actieve object. Vervolgens bepaalt de ingebouwde code van het object of een toetsaanslag een handeling vertegenwoordigt of een teken dat moet worden weergegeven in een veld. Vervolgens wordt de bijbehorende **action**- of **keyChar**-methode (zie verderop in deze lijst) aangeroepen.

Stel dat een veldobject in een tabelobject actief is en dat de gebruiker op *Enter* drukt. De toetsaanslag activeert **keyPhysical**. Deze methode interpreteert de toetsaanslag als een verzoek om een handeling en vertaalt de toetsaanslag naar **action(FieldEnter)**. Hierdoor wordt weer de ingebouwde **action**-methode geactiveerd. Als de gebruiker echter op *K* drukt, activeert de toetsaanslag **keyPhysical** en wordt de aanslag geïnterpreteerd als een teken. Hierdoor wordt de **keyChar**-methode geactiveerd.

Het actiepakket voor **keyPhysical** bevat informatie over het Windows WM_KEYDOWN-bericht en een optionele WM_CHAR. Het actiepakket geeft dus zowel de virtuele toetscode als het ANSI-teken. U kunt code koppelen aan deze methode, maar dat is niet verstandig, tenzij u speciale tekens wilt verwerken. Als u bijvoorbeeld de toets *F9*

expliciet wilt onderscheppen, in plaats van **action(DataToggleEdit)** te verwerken, is dit de methode die u moet gebruiken.

*keyChar* **keyChar** wordt aangeroepen als een **keyPhysical** niet wordt geïnterpreteerd door Paradox. Voor elke **keyPhysical** die *niet* wordt vertaald naar een handeling (zie **action** in deze lijst), wordt **keyChar** dus aangeroepen. **keyChar** gaat eerst naar het formulier en wordt door het formulier doorgestuurd naar het actieve object.

Tijdens de bewerking van een veld vergrendelt het systeem het record voordat het eerste teken wordt ingevoegd.

Als een knop een **keyChar** ontvangt die overeenkomt met een druk op *Spatiebalk* (bijvoorbeeld **keyChar(VK_SPACE)**), wordt de **pushButton**-methode van de knop aangeroepen.

*menuAction* **menuAction** wordt aangeroepen als de gebruiker een opdracht in een menu kiest (of klikt op een knop op de TurboBalk die een menuhandeling uitvoert). **menuAction** gaat eerst naar het formulier en wordt door het formulier doorgestuurd naar het actieve object. Zie Hoofdstuk 12 voor meer informatie.

*error* **error** wordt aangeroepen als een fout optreedt. Objecten (behalve formulieren) geven fouten standaard door aan de insluitende objecten. U kunt code koppelen aan de standaardmethode om ervoor te zorgen dat een object fouten behandelt, doorgeeft of beide. Raadpleeg Hoofdstuk 19 voor meer informatie over fouten en de behandeling van fouten.

*status* **status** wordt aangeroepen voordat een bericht wordt weergegeven in een van de gebieden op de statusbalk. U kunt bijvoorbeeld code koppelen aan de ingebouwde **status**-methode om berichten door te sturen naar andere gebieden of om de tekst van het bericht te veranderen. Zie Hoofdstuk 12 voor meer informatie.

*action* **action** wordt aangeroepen als **keyPhysical** een toetsaanslag vertaalt naar een handeling, als **menuAction** een menukeuze vertaalt naar een handeling en als andere methodes een handeling willen laten uitvoeren. **action** gaat eerst naar het formulier en wordt door het formulier doorgestuurd naar het doelobject. Een druk op *F2* in een veld activeert bijvoorbeeld standaard **action(EditToggleFieldView)** nadat de **keyPhysical**-methode is uitgevoerd. Een klik op de stuurknop activeert **action(DataNextRecord)** nadat de **menuAction**-methode is uitgevoerd.

**action** is een zeer belangrijke methode. Deze methode wordt gedetailleerd besproken in Hoofdstuk 13.

## Speciale ingebouwde methodes

Sommige objecten hebben aanvullende ingebouwde methodes, die in deze paragraaf worden besproken.

### pushButton

Alleen gedefinieerd voor knopobjecten en velden die worden weergegeven als keuzelijsten. **pushButton** wordt aangeroepen als de gebruiker de muis loslaat boven een knop. Deze methode wordt niet direct door het formulier aangeroepen, maar door de standaard **mouseUp**-methode voor knoppen. U kunt deze methode ook direct aanroepen om de normale handeling te laten uitvoeren die hoort bij de druk op een knopobject.

Het uiterlijk van een knop verandert standaard als op de knop wordt geklikt. Een drukknop wordt bijvoorbeeld ingedrukt en weer losgelaten; een aankruisvak wordt geselecteerd of gedeselecteerd; keuzeknoppen worden ingedrukt of losgelaten . Als u op een knop klikt, wordt de focus naar de knop verplaatst (tenzij het kenmerk Tab Stop van de knop is ingesteld op False).

Als het kenmerk Tab Stop van een knop is ingesteld op True en de knop het actieve object is, zijn er twee manieren om met het toetsenbord de **pushButton**-methode van de knop te activeren:

- Druk op *Spatiebalk*. De knop behoudt de focus.
- Druk op *Enter*. De focus wordt verplaatst naar het volgende object in de tab-volgorde.

### changeValue

Alleen gedefinieerd voor veldobjecten. **changeValue** vraagt toestemming om de waarde van een veld te veranderen. **changeValue** wordt aangeroepen *voordat* de waarde wordt opgeslagen, dus u kunt de waarde controleren en er iets mee doen (bijvoorbeeld aanvullende validiteitscontroles uitvoeren). **changeValue** wordt niet aangeroepen als iemand een waarde verandert via een netwerk of een opzoekdefinitie voor alle overeenkomende velden.

De volgende instructie activeert de **changeValue**-methode van *Aantal*, zelfs als *Aantal* al 10 is. De activering treedt direct op. Er wordt niet gewacht totdat de uitvoering van de methode is voltooid.

```
Aantal = 10
```

### newValue

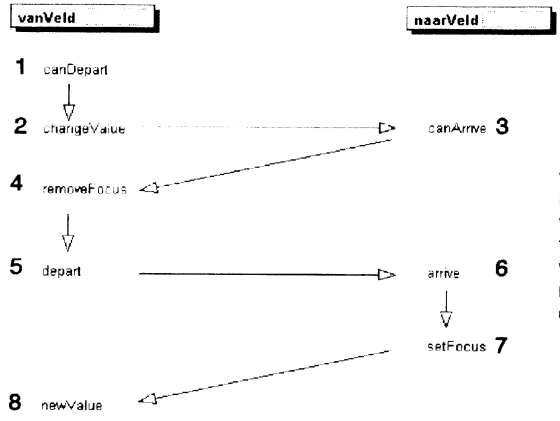
Alleen gedefinieerd voor veldobjecten. **newValue** wordt aangeroepen om te rapporteren dat een veldobject een nieuwe waarde heeft. Als u bijvoorbeeld door een tabel schuift en een nieuw record bereikt, wordt **newValue** geactiveerd. Als een veld wordt weergegeven als keuzeknoppen, wordt **newValue** aangeroepen als u op een knop klikt. Als u alleen in een veldobject typt, wordt **newValue** niet geactiveerd (wel wordt het kenmerk Touched ingesteld op True). **changeValue** wordt in ieder geval pas aangeroepen als u probeert het veldobject te verlaten of op een andere manier probeert veranderingen door te voeren. De **open**-methode van een formulier activeert ook **newValue** voor alle veldobjecten op het formulier.

*Veldobjecten met en zonder label*

Afbeelding B-3 toont de volgorde waarin ingebouwde methodes worden uitgevoerd als u van het ene veld (met of zonder label) naar het andere gaat nadat u een waarde hebt bewerkt. Het veldobject waar u vandaan komt, heet *vanVeld*. Het object waar u naartoe gaat, heet *naarVeld*.

***Opmerking*** **newValue** wordt aangeroepen als Paradox de waarde van het veldobject wil bijwerken (in dit geval, de weergave bijwerken). Aanroepen van **newValue** maken geen deel uit van de **canDepart**-sequentie. Daarom verschijnt **newValue** na alle andere in Afbeelding B-2. **newValue** is echter niet per se de laatste methode die wordt uitgevoerd.

Afbeelding B-2 Verplaatsen tussen veldobjecten als een waarde is gewijzigd

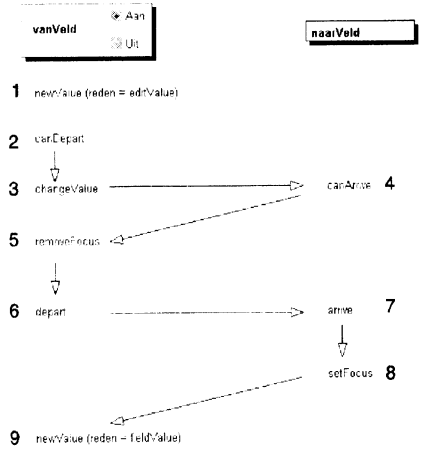



In deze afbeelding wordt de volgorde getoond waarin ingebouwde methodes worden uitgevoerd als de focus tussen twee veldobjecten wordt verplaatst nadat de waarde in het eerste veldobject is gewijzigd.

*Keuzeknoppen en lijsten*

Afbeelding B-3 toont de volgorde waarin ingebouwde methodes worden uitgevoerd als u van het ene veldobject (keuzeknoppen, een lijst of een afrollijst) naar het andere gaat nadat u een waarde hebt bewerkt. De volgorde is hetzelfde als voor normale velden, met uitzondering van een aanvullende aanroep van **newValue** als u een keuzeknop of een element in een lijst kiest. De tabel toont ook de ValueReason-constante voor elke **newValue**. Het veldobject waar u vandaan komt, heet *vanVeld*. Het object waar u naartoe gaat, heet *naarVeld*.

Afbeelding B-3 Verplaatsen van een keuzeknop of een lijst naar een veldobject




De eerste **newValue**-methode wordt uitgevoerd als u op de keuzeknop klikt (of als u een item uit de lijst kiest). Alle andere methodes worden uitgevoerd als u de verplaatsing start.

## changeValue gebruiken met veldobjecten

De ingebouwde code voor **changeValue** voert de wijzigingen in de waarde door. Totdat de ingebouwde code wordt uitgevoerd, gebruikt Paradox de oude, ongewijzigde waarde. Stel dat een veldobject een waarde van 10 heeft, dat u naar het veld gaat en de waarde 23 invoert. Als u vervolgens het veldobject verlaat, activeert u de **changeValue**-methode van het object, waaraan de volgende code is gekoppeld:

```
method changeValue(var eventInfo ValueEvent)
   msgInfo("voor de verandering", self.value) ; geeft 10 weer
   doDefault
   msgInfo("na de verandering", self.value)   ; geeft 23 weer
endmethod
```

Als deze methode wordt uitgevoerd, toont het eerste dialoogvenster de oude waarde, 10, omdat de ingebouwde code nog niet is uitgevoerd. Vervolgens voert de aanroep van **doDefault** de ingebouwde code uit. Deze code voert de veranderde waarde door. Het tweede dialoogvenster toont de veranderde waarde.

In de **changeValue**-methode van een object kunt u de ValueEvent-methode **newValue** (niet te verwarren met de ingebouwde **newValue**-methode, die eerder is besproken) gebruiken om de nieuwe waarde op te vragen voordat de ingebouwde code wordt uitgevoerd. Stel dat, zoals in het vorige voorbeeld, een veldobject een waarde van 10 heeft, dat u naar het veld gaat, de waarde 23 invoert en de **changeValue**-methode activeert, waaraan de volgende code is gekoppeld:

```
method changeValue(var eventInfo ValueEvent)
   msgInfo("voor de verandering", self.value)       ; geeft 10 weer
   msgInfo("de nieuwe waarde", eventInfo.newValue()) ; geeft 23 weer
   doDefault
```

```
   msgInfo("na de verandering", self.value)          ; geeft 23 weer
endmethod
```

Het eerste dialoogvenster geeft de oude, onveranderde waarde weer. Het tweede dialoogvenster roept **eventInfo.newValue** aan om de nieuwe waarde weer te geven—deze waarde is echter nog niet doorgevoerd. De aanroep van **doDefault** voert de ingebouwde code uit, die de wijziging doorvoert, en het derde dialoogvenster geeft de veranderde waarde weer.

Raadpleeg Hoofdstuk 12 voor meer informatie over het gebruik van **changeValue**.

# Standaardgedrag bepalen

De volgende elementaire taalonderdelen bepalen wanneer (en of) de ingebouwde code wordt uitgevoerd:

- **doDefault** voert de ingebouwde code meteen uit en voorkomt dat de ingebouwde code opnieuw wordt uitgevoerd op het eind van de methode.
- **disableDefault** blokkeert de uitvoering van de ingebouwde code.
- **enableDefault** staat toe dat de ingebouwde code op het eind van een methode wordt uitgevoerd.
- **passEvent** geeft de actie door aan het insluitend object van een object, ongeacht of het object de actie heeft behandeld. **passEvent** heeft geen effect op de uitvoering van de ingebouwde code op het eind van de methode.

***Belangrijk*** U kunt deze elementen alleen gebruiken in code die is gekoppeld aan ingebouwde methodes en niet in eigen methodes of procedures.

De ingebouwde **keyChar**-methode van een nieuw veldobject luidt bijvoorbeeld als volgt:

```
method keyChar(var eventInfo KeyEvent)

endmethod
```

Tekens verschijnen als u deze in dit veldobject typt, omdat de ingebouwde code vlak voor het sleutelwoord **endMethod** impliciet wordt uitgevoerd.

## doDefault

Als u de volgende code aan de methode koppelt, zou u misschien verwachten dat elk teken dat u typt, tweemaal verschijnt: eenmaal voor uw expliciete aanroep en eenmaal voor de impliciete aanroep. De tekens verschijnen echter slechts eenmaal.

```
method keyChar(var eventInfo KeyEvent)
   doDefault
endMethod
```

Een expliciete aanroep van **doDefault** blokkeert de impliciete aanroep van de ingebouwde code.

***Belangrijk*** Hoewel de compiler meerdere aanroepen van **doDefault** toestaat, heeft alleen de eerste aanroep effect. De overige aanroepen worden genegeerd.

## disableDefault

U krijgt hetzelfde effect—namelijk dat de impliciete code niet wordt uitgevoerd—als u **disableDefault** of **enableDefault** gebruikt.

Als een sleutelwoord zich in een methode bevindt, maar niet wordt uitgevoerd, heeft dat geen invloed op de impliciete aanroep. Bijvoorbeeld:

```
method action(var eventInfo ActionEvent)
   if eventInfo.id() = DataPostRecord then
      if testWaarde > 10 then
         message("De waarde is te groot.")
         disableDefault ; blokkeer de ingebouwde code
      else
         message("De waarde is OK.") ; voer ingebouwde code impliciet uit
      endIf
   endIf
endMethod
```

In het vorige voorbeeld wordt het sleutelwoord **disableDefault** alleen uitgevoerd en wordt de ingebouwde code alleen uitgeschakeld als de waarde van *testWaarde* groter is dan 10. Anders wordt de ingebouwde code geactiveerd en op het eind van de methode uitgevoerd.

Als u bepaalde gevallen zelf wilt behandelen en de overige door ObjectPAL wilt laten behandelen, roept u **disableDefault** aan voor het **switch**-blok. Daarna roept u **enableDefault** of **doDefault** aan in het hoofddeel van het blok. De volgende code bijvoorbeeld staat de uitvoering van de standaardcode alleen toe als de ID van de handeling DataNextRecord of DataPriorRecord is:

```
method action(var eventInfo ActionEvent)
   disableDefault ; Uitgangspunt: geen standaard

   ; Maar als ID van handeling DataNextRecord of DataPriorRecordis,
   ; wordt standaard uitgevoerd.

   switch eventInfo.Id()
      case DataNextRecord  : enableDefault
      case DataPriorRecord : enableDefault
   endSwitch
endMethod
```

Deze aanpak is handig voor de verwerking van menukeuzen. Bijvoorbeeld:

```
method menuAction(var eventInfo MenuEvent)
   var deKeuze String endVar
```

```
   disableDefault
   deKeuze = eventInfo.menuChoice()
   switch
     case deKeuze = "Openen" : doOpen() ; eigen methode, standaard niet
uitvoeren
     case deKeuze = "Nieuw"  : doNew()  ; eigen methode, standaard niet
uitvoeren
     otherwise               : enableDefault
   endSwitch
endMethod
```

***Belangrijk*** Een **return**-instructie in een ingebouwde methode voert de ingebouwde code direct vóór de **return** uit, tenzij u **disableDefault** aanroept om de ingebouwde code te blokkeren. De volgende code roept bijvoorbeeld **disableDefault** aan om te voorkomen dat het veldobject het teken X weergeeft:

```
method keyChar(var eventInfo KeyEvent)
   if eventInfo.char() = "X" then
      disableDefault
      return
   endIf
endmethod
```

***Belangrijk*** Gebruik **disableDefault** niet in methodes voor interne acties. Als u de standaardcode voor deze methodes blokkeert, kan dat leiden tot onverwachte resultaten. Gebruik in plaats daarvan **setErrorCode** om een niet-nul foutwaarde in te stellen. De standaardcode controleert de foutwaarde als onderdeel van de normale uitvoering; een niet-nul foutwaarde duidt op een fout en de standaardcode houdt hier rekening mee.

Stel dat de volgende code is gekoppeld aan de ingebouwde **canDepart**-methode van een veldobject. Als deze code wordt uitgevoerd, wordt de datumwaarde in het veldobject vergeleken met de datum van vandaag. Als de datum in het veldobject later is dan de datum van vandaag, verschijnt een dialoogvenster dat de gebruiker op de hoogte stelt en gebruikt de aanroep van **setErrorCode** een voorgedefinieerde ObjectPAL-constante (CanNotDepart) om een niet-nul foutwaarde in te stellen. Als de standaardcode wordt uitgevoerd, "ziet" de code deze waarde en wordt voorkomen dat de cursor het veldobject verlaat.

```
method canDepart(var eventInfo MoveEvent)
   if Date(Self.value) > today() then
      msgStop("Stop", "Deze datum ligt in de toekomst.")
      eventInfo.setErrorCode(CanNotDepart)
   endIf
endmethod
```

Een foutwaarde verwerken is *niet* hetzelfde als de standaardcode uitschakelen. Raadpleeg Hoofdstuk 12 voor meer informatie en voorbeelden.

## enableDefault

**enableDefault** staat toe dat de ingebouwde code wordt uitgevoerd op het eind van een methode. Het verschil tussen **doDefault** en **enableDefault** is belangrijk: **doDefault** voert de ingebouwde code direct uit en **enableDefault** zet een vlag, zodat de ingebouwde code op het eind van de methode wordt uitgevoerd. Als u bijvoorbeeld de volgende code aan een veldobject koppelt en een teken in het veld typt, wacht Paradox drie seconden, klinkt er een pieptoon en verschijnt het teken:

```
method keyChar(var eventInfo KeyEvent)
   enableDefault
   sleep(3000)
   beep()
endMethod
```

De volgende code zorgt er daarentegen voor dat Paradox het teken weergeeft vóór de pauze en de pieptoon:

```
method keyChar(var eventInfo KeyEvent)
   doDefault
   sleep(3000)
   beep()
endMethod
```

***Belangrijk*** Een aanroep van **doDefault** zet een vlag die voorkomt dat de ingebouwde code op het eind van de methode wordt uitgevoerd.

U kunt deze sleutelwoorden overal in een ingebouwde methode opnemen, maar houd er rekening mee dat het gedrag van een object afhankelijk kan zijn van de plaats waar de sleutelwoorden worden gebruikt. Stel dat de volgende code is gekoppeld aan een ongedefinieerd veld:

```
method keyChar(var eventInfo KeyEvent)
   eventInfo.setChar("K")
   doDefault
endMethod
```

Als u deze methode start en tekens in het veld typt, verschijnt de letter K ongeacht wat u typt, omdat de methode het teken instelt op K en *daarna* de ingebouwde code aanroept die het teken in het veld weergeeft. Als u de volgorde van de instructies omdraait, krijgt u een ander resultaat:

```
method keyChar(var eventInfo KeyEvent)
   doDefault
   eventInfo.setChar("K")
endMethod
```

In het vorige voorbeeld voert de aanroep van **doDefault** de ingebouwde code uit voordat het teken wordt ingesteld op K, zodat het veld zich normaal gedraagt.

## passEvent

Een **passEvent**-instructie zegt eigenlijk: "Onderbreek de uitvoering en stuur de informatie over deze actie naar mijn insluitend object". Dit

heeft geen effect op de ingebouwde code. Als de ingebouwde code impliciet of met **enableDefault** is geactiveerd, wordt deze op het eind van de methode uitgevoerd. Als **passEvent** wordt aangeroepen en de ingebouwde code is uitgeschakeld door **disableDefault** of wordt aangeroepen door **doDefault**, wordt deze niet uitgevoerd op het eind van de methode.

Een aanroep van **passEvent** onderbreekt de uitvoering van de aanroepende methode en activeert direct de juiste methode in het insluitende object van het aanroepende object. De aanroepende methode hervat de uitvoering standaard als de methode van het insluitend object is voltooid.

Als u wilt zien hoe dit werkt, volgt u deze stappen om een eenvoudig formulier te maken en te starten:

1. Maak een nieuw formulier.
2. Plaats een groot kader op de pagina.
3. Plaats een veldobject in het kader, zodat het kader het veldobject insluit.
4. Koppel de volgende code aan de ingebouwde **keyChar**-methode van het veldobject:

```
method keyChar(var eventInfo KeyEvent)
   msgInfo("keyChar", "Aanroep van keyChar-methode van insluitend object.")
   passEvent
   msgInfo("keyChar", "We zijn weer thuis.")
endmethod
```

5. Koppel de volgende code aan de ingebouwde **keyChar**-methode van het kader:

```
method keyChar(var eventInfo KeyEvent)
   self.color = self.color + 222
   sleep(1000)
endmethod
```

6. Start het formulier en typ een teken in het veldobject.

Als u dit formulier start en een teken in het veldobject typt, gebeurt het volgende: eerst opent de code die is gekoppeld aan de ingebouwde **keyChar**-methode van het veldobject een dialoogvenster. Vervolgens activeert de aanroep van **passEvent** de **keyChar**-methode van het insluitend object van het veldobject, het kader. De **keyChar**-methode van het veldobject onderbreekt de uitvoering. Als de **keyChar**-methode van het kader wordt uitgevoerd, verandert de kleur van het kader en pauzeert Paradox een seconde. Als de **keyChar**-methode van het kader is voltooid, hervat de **keyChar**-methode van het veldobject de uitvoering en verschijnt er weer een dialoogvenster.

Raadpleeg de onderwerpen over **doDefault**, **disableDefault**, **enableDefault** en **passEvent** in het online Helpsysteem van ObjectPAL.

# Ingebouwde objectvariabelen

Naast ingebouwde methodes kent ObjectPAL ingebouwde objectvariabelen:

*Self* *Self* verwijst naar het object waaraan de code is gekoppeld die op het moment wordt uitgevoerd.Als de volgende instructie bijvoorbeeld wordt uitgevoerd in de **mouseEnter**-methode die is gekoppeld aan *hetKader*, verwijst *Self* naar *hetKader*.

```
self.color = Red
```

Stel echter dat een methode die is gekoppeld aan *hetKader* een eigen methode aanroept met de naam **veranderKleur**, die is gekoppeld aan de pagina. Stel dat de code van **veranderKleur** als volgt luidt:

```
method veranderKleur()
   self.color = Blue
endMethod
```

Als de code die is gekoppeld aan *hetKader*, **veranderKleur** aanroept, wordt de pagina blauw, niet *hetKader*. Waarom? Omdat *Self* verwijst naar het object waaraan de code is gekoppeld, ongeacht welk object de code heeft aangeroepen. In dit geval is de code gekoppeld aan de pagina. De enige uitzondering op deze regel treedt op wanneer *Self* verschijnt in een instructie in een bibliotheek. In dat geval verwijst *Self* naar het object dat de bibliotheekroutine heeft aangeroepen.

Als een actie optreedt, kunnen *Self* en **eventInfo.getTarget** naar hetzelfde object verwijzen, maar terwijl acties omhoogborrelen door de hiërarchie van ingesloten objecten, blijft het doel ongewijzigd, terwijl *Self* verandert en verwijst naar het object dat de methode uitvoert.

***Opmerking*** *Self* verwijst altijd naar een UIObject, niet naar de waarde van het object of naar de naam van het object.

*Container* *Container* is het object dat *Self* insluit. Stel dat een kader een knop bevat en dat de **pushButton**-methode van de knop als volgt luidt:

```
container.color = Red
```

Als deze code wordt uitgevoerd, wordt het kader rood, omdat het kader de knop bevat en de knop de code uitvoert.

*Active* *Active* is het huidige actieve object, het laatste object dat een **moveTo** heeft ontvangen (zie de paragraaf UIObject in Hoofdstuk 13 voor

meer informatie over de **moveTo**-methode). Het actieve object is normaal gesproken gemarkeerd.

Als een actief object zo is ingesteld dat het reageert op toetsenbordacties, heeft dit object de *focus*. Ook als de focus van een object wordt verwijderd (bijvoorbeeld omdat een ander formulier wordt geactiveerd), verwijst *Active* nog naar dat object. Alleen als het object wordt verlaten, wordt *Active* ingesteld op het nieuwe object. Onderschat het belang van *Active* niet. Algemene routines kunnen zo worden geschreven dat deze op het actieve object betrekking hebben, zonder te weten welk object dat is. Stel dat een formulier twee tabelframes bevat die beide zijn verbonden met een andere tabel. De volgende instructie heeft automatisch betrekking op het actieve tabelframe:

```
active.action(DataNextRecord)
```

*Subject* *Subject* bepaalt op welk object een eigen methode betrekking heeft. Stel dat een pagina op een formulier een eigen methode met de naam **bepaalKleur** heeft en dat de code voor **bepaalKleur** als volgt luidt:

```
method bepaalKleur()
   subject.color = red
endmethod
```

Elk object op die pagina kan de onderstaande aanroep doen om het object *eenObject* rood te maken. Als **bepaalKleur** wordt uitgevoerd, wordt *Subject* vervangen door *eenObject*.

```
eenObject.bepaalKleur()
```

Raadpleeg Hoofdstuk 13 voor meer informatie en voorbeelden.

*LastMouseClicked* *LastMouseClicked* verwijst naar het object dat het laatst een **mouseDown** heeft ontvangen. *LastMouseClicked* wordt opnieuw ingesteld als de muisknop wordt losgelaten, maar eerst krijgt het object de gelegenheid een **mouseUp** uit te voeren.

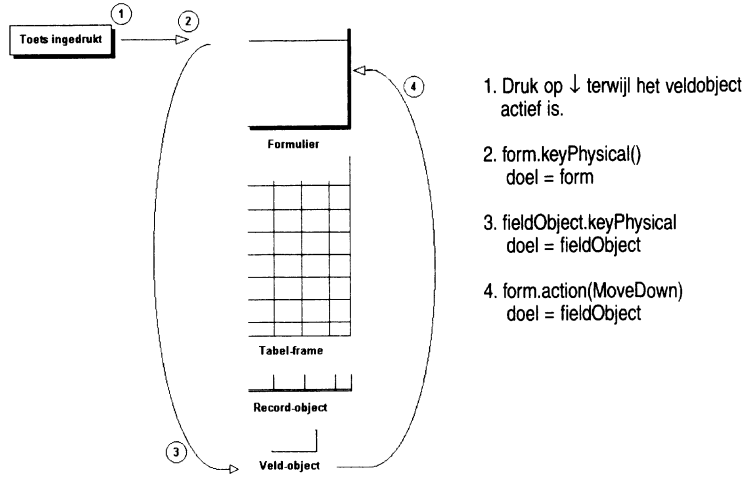
*LastMouseRightClicked* *LastMouseRightClicked* is hetzelfde als *LastMouseClicked*, maar voor de rechter muisknop.

# Geavanceerd onderwerp: voorbeeld van het verloop van methode-aanroepen

***Opmerking*** In deze paragraaf wordt technische informatie gegeven voor ervaren programmeurs. U hebt deze informatie niet nodig om ObjectPAL effectief te kunnen gebruiken.

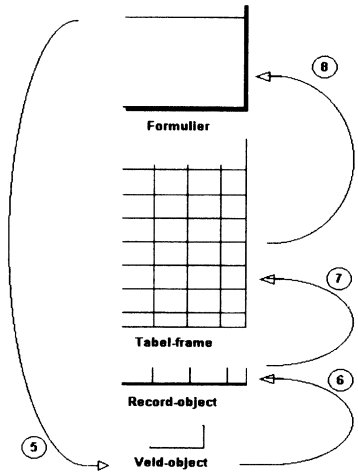
Hier volgt een voorbeeld van een verzameling handelingen. Stel dat u zojuist tekens hebt getypt in een veldobject in de derde rij van een tabelframe en dat u op ↓ hebt gedrukt. In Afbeelding B-4, B-5 en B-6 ziet u het verloop van de acties en de methodes:

Afbeelding B-4 Stappen 1 tot en met 4



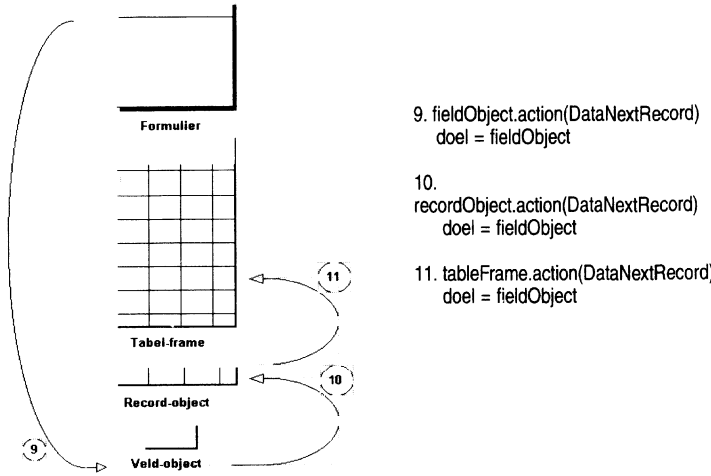

1. Druk op ↓.
2. Paradox maakt een KeyEvent-actiepakket en roept de **keyPhysical**-methode van het formulier aan. Het formulier is het doel.
3. In het KeyEvent-actiepakket stelt het formulier het doel in op het veldobject (omdat het actief is) en de actie wordt doorgestuurd naar de **keyPhysical**-methode van het veldobject.
4. De standaard **keyPhysical**-methode van het veldobject vertaalt de toetsaanslag naar een handeling. De methode maakt een ActionEvent-actiepakket met de handelingsconstante MoveDown en de **action**-methode van het formulier wordt aangeroepen. Met andere woorden, naar het formulier wordt **action(MoveDown)** gestuurd en het veldobject is het doel.

Afbeelding B-5 Stappen 5 tot en met 8




5. De **action**-methode van het formulier stuurt de actie naar de **action**-methode van het veldobject.
6. De **action**-methode van het veldobject kent MoveDown niet en laat de actie dus één niveau omhoogborrelen naar het insluitende recordobject.
7. De **action**-methode van het recordobject kent MoveDown ook niet en laat de actie dus omhoogborrelen naar het tabelobject.
8. Het tabelobject kent MoveDown, vertaalt MoveDown naar **action(DataNextRecord)** en roept de **action**-methode van het formulier aan, weer met het veldobject als doel.

Afbeelding B-6 Stappen 9 tot en met 11




9. Het formulier stuurt de actie met de constante DataNextRecord naar de **action**-methode van het veldobject.

10. De **action**-methode van het veldobject laat de DataNextRecord weer omhoogborrelen naar het record.

11. Het record laat het omhoogborrelen naar het tabelframe.

Wat daarna gebeurt, is afhankelijk van de grootte van het tabelframe. Als het een record bevat onder het huidige record, gaat Paradox gewoon naar het overeenkomende veldobject in dat record. Als het tabelframe niet meer records weergeeft (met andere woorden, als het huidige record het laatste weergegeven record is) en Paradox in het tabelframe moet schuiven, behandelt het tabelframe de DataNextRecord als volgt:

- Er wordt een **moveTo** naar het tabelframe gegenereerd, waardoor een **canDepart**, een **removeFocus** en een **depart** optreden voor het veldobject en het recordobject. Hierdoor wordt de waarde van het veldobject doorgevoerd in het record (maar niet in de onderliggende tabel) en verdwijnt de selectiemarkering.
- Als het record is gewijzigd, roept de **depart**-methode van het record de **action**-methode van het tabelframe aan met de constante DataUnlockRecord. Standaard wordt hierdoor geprobeerd het record door te voeren in de onderliggende tabel. Als de database-engine het record accepteert, wordt de code 0 teruggegeven, wat duidt op succes.

- Het tabelframe vraagt de database-engine één record verder te gaan.
- Als dat lukt, bepaalt het tabelframe welk record zich op het moment op het scherm bevindt.
- Het tabelframe genereert een **moveTo** naar het volgende veldobject, waardoor een **canArrive**, een **arrive** en een **setFocus** worden aangeroepen voor het nieuwe record en de veldobjecten.

Dit lijken misschien veel stappen, maar het gebeurt erg snel. Het voordeel voor u, als programmeur, is dat u beschikt over veel flexibiliteit en een hoge mate van controle.

APPENDIX C

# Kenmerken

Deze appendix geeft een overzicht van de kenmerken en kenmerkwaarden voor alle UIObjecten. Een dichte punt (●) duidt op een lezen-schrijven kenmerk en een open punt (○) duidt op een alleen-lezen kenmerk. Deze informatie is ook online beschikbaar. Als u de lijst wilt bekijken, opent u een venster van de ObjectPAL-Editor en kiest u 'Taal | Kenmerken'. Kies vervolgens een objectnaam en een kenmerk.

Table C-1 UIObjectkenmerken

| | Band | Bitmap | Box | Button | Cell | Crosstab | EditRegion | Ellipse | Field | Form | Graph | Group | Header | Line | List | Multirecord | OLE | Page | Record | TableFrame | TableView | Text | TVData | TVHeading |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alignment | | | | | | | ● | | ● | | | | | | | | | | | | | ● | ● | ● |
| Arrived | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | | ○ | | |
| BlankRecord | | | | | | | | | ○ | ○ | | | | | | ○ | | | ○ | ○ | ○ | | ○ | |
| Border | | | | | | | | | | ● | | | | | | | | | | | | | | |
| Breakable | ● | | | | | | ● | | | | | | | | | | | | | | | | | |
| ButtonType | | | | ● | | | | | | | | | | | | | | | | | | | | |
| Caption | | | | | | | | | | ● | | | | | | | | | | | | | | |
| CenterLabel | | | | ● | | | | | | | | | | | | | | | | | | | | |
| Class | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | | ○ | ○ | ○ |
| Color | | | ● | | ● | ● | ● | ● | ● | | ● | | ● | ● | | ● | | ● | ● | ● | ● | ● | ● | ● |
| CompleteDisplay | | | | | | | ● | | ● | | | | | | | | | | | | | | ● | |
| ContainerName | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | | ○ | | |
| ControlMenu | | | | | | | | | | ● | | | | | | | | | | | | | | |

| | Band | Bitmap | Box | Button | Cell | Crosstab | EditRegion | Ellipse | Field | Form | Graph | Group | Header | Line | List | Multirecord | OLE | Page | Record | TableFrame | TableView | Text | TVData | TVHeading |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CurrentRecordMarker.Color | | | | | | | | | | | | | | | | | | | | | ● | | | |
| CurrentRecordMarker.LineStyle | | | | | | | | | | | | | | | | | | | | | ● | | | |
| CurrentRecordMarker.Show | | | | | | | | | | | | | | | | | | | | | ● | | | |
| CursorColumn | | | | | | | | | ● | | | | | | | | | | | | | ● | | |
| CursorLine | | | | | | | | | ● | | | | | | | | | | | | | ● | | |
| CursorPos | | | | | | | | | ● | | | | | | | | | | | | | ● | | |
| DataSource | | | | | | | | | | | | | | | ● | | | | | | | | | |
| DateFormat | | | | | | | | | | | | | | | | | | | | | | | ● | |
| Default | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| Deleted | | | | | | | | | ○ | ○ | | | | | | ○ | | | ○ | ○ | ○ | | ○ | |
| Design.ContainObjects | ● | ● | ● | ● | ● | | | ● | | | ● | | ● | | | | ● | ● | | | | ● | | |
| Design.PinHorizontal | ● | ● | ● | ● | ● | ● | ● | ● | ● | | ● | ● | ● | ● | | ● | ● | ● | | ● | | ● | | |
| Design.PinVertical | ● | ● | ● | ● | ● | ● | ● | ● | ● | | ● | ● | ● | ● | | ● | ● | ● | | ● | | ● | | |
| Design.SizeToFit | | ● | | | ● | | ● | | ● | | | | | | | | ● | ● | | ● | | | | |
| DesignSizing | | | | | | | | | | ● | | | | | | | | | | | | ● | | |
| DesktopForm | | | | | | | | | | ● | | | | | | | | | | | | | | |
| DialogForm | | | | | | | | | | ● | | | | | | | | | | | | | | |
| DialogFrame | | | | | | | | | | ● | | | | | | | | | | | | | | |
| DisplayType | | | | | | | ○ | | ● | | | | | | | | | | | | | | | |
| Editing | | | | | | | | | ○ | ○ | | | | | | | | | | | ○ | | | |
| End | | | | | | | | | | | | | | ● | | | | | | | | | | |
| FieldName | | | | | | | | | ● | | | | | | | | | | | | ○ | | ○ | |
| FieldNo | | | | | | | | | ○ | | | | | | | | | | | | ○ | | ○ | |
| FieldRights | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| FieldSize | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| FieldType | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| FieldUnits2 | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| FieldValid | | | | | | | | | ○ | | | | | | | | | | | | | | | |
| FieldView | | | | | | | | | | ○ | | | | | | | | | | | ○ | | | |
| First | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | | ○ | | |

Band
Bitmap
Box
Button
Cell
Crosstab
EditRegion
Ellipse
Field
Form
Graph
Group
Header
Line
List
Multirecord
OLE
Page
Record
TableFrame
TableView
Text
TVData
TVHeading
FlyAway
Focus
Font.Color
Font.Size
Font.Style
Font.Typeface
Format.DateFormat
Format.LogicalFormat
Format.NumberFormat
Format.TimeFormat
Format.TimeStampFormat
Frame.Color
Frame.Style
Frame.Thickness
FullName
FullSize
Grid.Color
Grid.GridStyle
Grid.RecordDivider
Gridlines.Color
Gridlines.ColumnLines
Gridlines.HeadingLines
Gridlines.LineStyle
Gridlines.RowLines
Gridlines.Spacing
HeadingHeight
Headings
HorizontalScrollBar
IndexField
Inserting

| | Band | Bitmap | Box | Button | Cell | Crosstab | EditRegion | Ellipse | Field | Form | Graph | Group | Header | Line | List | Multirecord | OLE | Page | Record | TableFrame | TableView | Text | TVData | TVHeading |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| KeyField | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| LabelText | | | | ● | | | | | ● | | | | | | | | | | | | | | | |
| Line.Color | | | | | | | | ● | | | | | | | | | | | | | | | | |
| Line.LineStyle | | | | | | | | ● | | | | | | | | | | | | | | | | |
| Line.Thickness | | | | | | | | ● | | | | | | | | | | | | | | | | |
| LineEnds | | | | | | | | | | | | | | ● | | | | | | | | | | |
| LineSpacing | | | | | | | | | | | | | | | | | | | | | | ● | | |
| LineStyle | | | | | | | | | | | | | | ● | | | | | | | | | | |
| LineType | | | | | | | | | | | | | | ● | | | | | | | | | | |
| List.Count | | | | | | | | | | | | | | | ● | | | | | | | | | |
| List.Selection | | | | | | | | | | | | | | | ● | | | | | | | | | |
| Locked | | | | | | | | | ○ | ○ | | | | | | ○ | | | ○ | ○ | ○ | | ○ | |
| LookupTable | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| LookupType | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| Magnification | | ● | | | | | ● | | ● | | | | | | | | ● | | | | | | ● | |
| Manager | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | | |
| MarkerPos | | | | | | | | | ● | | | | | | | | | | | | | ● | | |
| MaximizeButton | | | | | | | | | | ● | | | | | | | | | | | | | | |
| Maximum | | | | | | | | | ○ | | | | | | | | | | | | | | | |
| MemoView | | | | | | | | | | ○ | | | | | | | | | | | ○ | | | |
| MinimizeButton | | | | | | | | | | ● | | | | | | | | | | | | | | |
| Minimum | | | | | | | | | ○ | | | | | | | | | | | | | | ○ | |
| Modal | | | | | | | | | | ● | | | | | | | | | | | | | | |
| MouseActivate | | | | | | | | | | ● | | | | | | | | | | | | | | |
| NCols | | | | | | | | | | | | | | | | ● | | | | ● | ○ | | | |
| NRecords | | | | | | | | | ○ | ○ | | | | | | ○ | | | ○ | ○ | ○ | | ○ | |
| NRows | | | | | | | | | | | | | | | | ● | | | | ● | ○ | | | |
| Name | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● | ○ | ● | ● | ● |
| Next | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | | ○ | | |
| NoEcho | | | | | | | ● | | ● | | | | | | | | | | | | | | | |

Band
Bitmap
Box
Button
Cell
Crosstab
EditRegion
Ellipse
Field
Form
Graph
Group
Header
Line
List
Multirecord
OLE
Page
Record
TableFrame
TableView
Text
TVData
TVHeading
OverStrike
Owner
Pattern.Color
Pattern.Style
PersistView
Picture
Position
PrecedePageHeader
Prev
PrintOn1stPage
RasterOperation
Readonly
RecNo
Refresh
Required
RowHeight
RowNo
Scroll
SelectedText
SeqNo
Shrinkable
Size
SizeToFit
SortOrder
StandardMenu
Start
Style
TabStop
TableName
Text

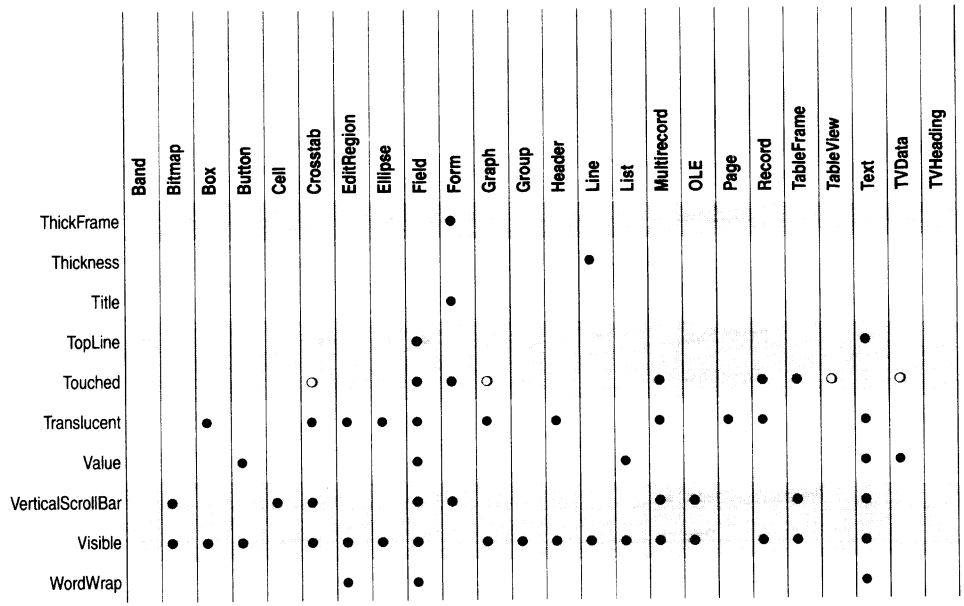



Table C-2 Kenmerken en hun waarden

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| Alignment | SmallInt | TextAlignCenter, TextAlignJustify, TextAlignLeft, TextAlignRight | Geeft de positie van tekst in verhouding tot een veldobject of een tekstobject. |
| Arrived | Logical | True, False | Geeft aan of een record actief is. |
| BlankRecord | Logical | True, False | Geeft aan of een record leeg is. |
| Border | Logical | True, False | Geeft aan of het venster van een formulier een kader heeft. |
| Breakable | Logical | True, False | Geeft aan of een object mag worden onderbroken door een pagina-afbreking. |
| ButtonType | SmallInt | CheckBoxType, PushButtonType, RadioButtonType | Geeft het weergavetype van een knop. |
| Caption | Logical | True, False | Geeft aan of een formulier een titelbalk heeft. |
| CenterLabel | Logical | True, False | Geeft aan of een label wordt gecentreerd binnen een knop. |
| Class | String | Band, Bitmap, Box, Button, Cell, Crosstab, EditRegion, Ellipse, Field, Form, Graph, Group, Header, Line, List, Multirecord, OLE, Page, Record, TableFrame, Text | Geeft de klasse van een UIObject terug. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de weergavekleur van een object. |
| CompleteDisplay | Logical | True, False | Geeft aan of de volledige inhoud van een veld wordt weergegeven. |
| ContainerName | String | Nvt | Geeft de naam van het insluitend object van een object. |
| CurrentRecordMarker.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de weergavekleur van het huidige record van een tabelweergave. |
| CurrentRecordMarker.LineStyle | SmallInt | DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine | Geeft de stijl van de lijn die het huidige record in een tabelweergave markeert. |
| CurrentRecordMarker.Show | Logical | True, False | Geeft aan of het huidige record in een tabelweergave wordt gemarkeerd. |
| CursorColumn | LongInt | Nvt | De horizontale positie van het invoegpunt in een veldobject, waarbij positie 0 links van het eerste teken ligt. |
| CursorLine | LongInt | Nvt | De verticale positie van het invoegpunt in een veldobject, waarbij de eerste regel regel 1 is. |
| CursorPos | LongInt | Nvt | De positie van het invoegpunt in een veldobject, in verhouding tot het eerste teken in het veld. De telling begint bij 0, de positie links van het eerste teken. |
| DataSource | String | Nvt | De naam van de tabel die elementen levert voor een lijst. |
| DateFormat | Nvt | Opmaakspecificatie | Geeft de datumopmaak van een veldobject. |
| Default | String | Nvt | De standaardwaarde van een veld. |
| DefineGroup | Logical | True, False | Geeft aan of een rapportzone een groep definieert. |
| Deleted | Logical | True, False | Geeft aan of een record in een dBASE-tabel is gemarkeerd als verwijderd. |
| Design.ContainObjects | Logical | True, False | Geeft aan of een object andere objecten kan insluiten. |
| Design.SizeToFit | Logical | True, False | Geeft aan of het object van grootte verandert om zich aan te passen aan de inhoud. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| DesignSizing | SmallInt | TextFixedSize, TextGrowOnly, TextSizeToFit | Geeft aan wat er tijdens de uitvoering met de grootte van het tekstvak kan gebeuren. |
| DesktopForm | Logical | True, False | Geeft aan of de menu's van een formulier worden gebruikt door andere formulieren op het Bureaublad. |
| DialogForm | Logical | True, False | Geeft aan of een formulier wordt geopend als een dialoogvenster. |
| DialogFrame | Logical | True, False | Als DialogFrame True is, evenals DialogForm en Border, dan heeft het formulier een normaal dialoogvensterkader. |
| DisplayType | SmallInt | BitmapField, CheckBoxField, ComboField, EditField, LabeledField, ListField, OleField, RadioButtonField | Geeft het weergavetype van een veldobject terug. |
| Editing | Logical | True, False | Geeft aan of voor een formulier de bewerkmodus is geactiveerd, of een veld actief is en wordt bewerkt. |
| End | Point | Nvt | Geeft de coördinaten van het einde van een regel. Zie ook: Start. |
| FieldName | String | Nvt | Geeft de naam van het veld waarmee een veldobject of een lijst is verbonden. |
| FieldNo | SmallInt | Nvt | Geeft de positie van een veld in een tabel, waarbij het eerste veld veld 1 is. |
| FieldRights | String | ReadOnly, ReadWrite, All | Geeft de veldrechten van de gebruiker. |
| FieldSize | SmallInt | Nvt | De grootte van het veld (string en dBASE-getallen). |
| FieldType | String | Nvt | Het gegevenstype van het veld. |
| FieldUnits2 | SmallInt | Nvt | Geeft het aantal decimale posities in een dBASE-getalveld. |
| FieldValid | Logical | True, False | Geeft aan of een veld de eigen waarde controleert. |
| FieldView | Logical | True, False | Geeft aan of voor een veld de veldweergave is geactiveerd. |
| First | String | Nvt | Geeft de naam terug van het eerste subobject in een insluitend object. |
| FitHeight | Logical | True, False | Geeft aan of een bewerkgebied verticaal groter wordt om zich aan te passen aan de tekst. |
| FitWidth | Logical | True, False | Geeft aan of een bewerkgebied of een kruistabulatie-cel horizontaal groter wordt om zich aan te passen aan de tekst. |
| FlyAway | Logical | True, False | Geeft aan of een record is verplaatst naar de gesorteerde positie in een tabel. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| Focus | Logical | True, False | Geeft aan of de ingebouwde **setFocus**-methode van een object is aangeroepen. Focus is False als de ingebouwde **removeFocus**-methode van een object is aangeroepen. |
| Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van de tekens die worden weergegeven in een veldobject of een tekstobject. |
| Font.Size | SmallInt | >0 | Geeft (in printerpunten) de grootte van de tekens die worden weergegeven in een veldobject of een tekstobject. |
| Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline | Geeft de stijl van de tekens die worden weergegeven in een veldobject of een tekstobject. |
| Font.Typeface | String | Afhankelijk van het systeem | Geeft de lettersoort van de tekens die worden weergegeven in een veldobject of een tekstobject. |
| Format.DateFormat | Nvt | Opmaakspecificatie | Geeft de opmaakspecificatie voor datumwaarden. |
| Format.LogicalFormat | Nvt | Opmaakspecificatie | Geeft de opmaak voor logische waarden. |
| Format.NumberFormat | Nvt | Opmaakspecificatie | Geeft de opmaak voor getalwaarden. |
| Format.TimeFormat | Nvt | Opmaakspecificatie | Geeft de opmaak voor tijdwaarden. |
| Format.TimeStampFormat | Nvt | Opmaakspecificatie | Geeft de opmaak voor tijdstempels. |
| Frame.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van het frame van een object. |
| Frame.Style | SmallInt | DashDotDotFrame, DashDotFrame, DashedFrame, DottedFrame, DoubleFrame, Inside3DFrame, NoFrame, Outside3DFrame, ShadowFrame, SolidFrame, WideInsideDoubleFrame, WideOutsideDoubleFrame | Geeft de stijl van het frame van een object. |
| Frame.Thickness | SmallInt | Nvt | Geeft de dikte in pixels van het frame van een object. |
| FullName | String | Nvt | Geeft de volledige naam (inclusief het pad van insluitende objecten) van een object op een formulier terug. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| FullSize | Point | Nvt | In schuivend object, geeft de werkelijke grootte als u het hele object zou kunnen zien. |
| Grid.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van het raster in een tabelframe. |
| Grid.GridStyle | SmallInt | tfSingleLine, tfDoubleLine, tfTripleLine, tf3D, tfNoGrid | Geeft de stijl van rasterlijnen in een tabelframe. |
| Grid.RecordDivider | Logical | True, False | Geeft aan of scheidingslijnen worden weergegeven tussen records in een tabelframe. |
| GridLines.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van rasterlijnen in een tabelvenster. |
| GridLines.ColumnLines | Logical | True, False | Geeft aan of kolomlijnen worden weergegeven in een tabelvenster. |
| GridLines.HeadingLines | Logical | True, False | Geeft aan of koplijnen worden weergegeven in een tabelvenster. |
| GridLines.LineStyle | SmallInt | DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine | Geeft de stijl van rasterlijnen in een tabelvenster. |
| GridLines.RowLines | Logical | True, False | Geeft aan of rasterlijnen worden weergegeven in een tabelvenster. |
| GridLines.Spacing | SmallInt | TextSingleSpacing, TextDoubleSpacing, TextTripleSpacing | Geeft de afstand tussen rasterlijnen in een tabelvenster. |
| Heading.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van de kop van een tabelvenster. |
| Heading.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van tekens in de kop van een tabelvenster. |
| Heading.Font.Size | SmallInt | Afhankelijk van het systeem | Geeft de grootte van tekens in de kop van een tabelvenster. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| Heading.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline | Geeft de stijl van tekens in de kop van een tabelvenster. |
| Heading.Typeface | String | Afhankelijk van het systeem | Geeft de lettersoort van tekens in de kop van een tabelvenster. |
| Heading.Justification | SmallInt | TextAlignTop, TextAlignBottom, TextAlignVCenter, TextAlignLeft, TextAlignRight, TextAlignCenter | Geeft de uitlijning van tekens in de kop van een tabelvenster. |
| Headings | String | "GroupOnly", "PageAndGroup" | Geeft aan welke rapportkopregels worden afgedrukt. |
| HorizontalScrollBar | Logical | True, False | Geeft aan of een tabelframe een horizontale schuifbalk heeft. |
| IndexField | Logical | True, False | Geeft aan of een veldobject is verbonden met een geïndexeerd veld in een tabel. |
| Inserting | Logical | True, False | Geeft True terug als een record ergens op een formulier wordt ingevoegd. |
| Justification | SmallInt | TextAlignTop, TextAlignBottom, TextAlignVCenter, TextAlignLeft, TextAlignRight, TextAlignCenter | Geeft de uitlijning van gegevens in een tabelvenster. |
| KeyField | Logical | True, False | Geeft aan of een veldobject is verbonden met een sleutelveld in een tabel. |
| LabelText | String | Nvt | Geeft de tekst die wordt weergegeven in het label van een knop. |
| Line.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van een lijn voor een ellips. |
| Line.LineStyle | SmallInt | DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine | Geeft de stijl van een lijn voor een ellips. |
| Line.Thickness | SmallInt | Nvt | Geeft de dikte van een lijn voor een ellips. |
| LineEnds | SmallInt | NoArrow, OnBothEnds, OnOneEnd | Geeft aan of (en waar) pijlen moeten worden geplaatst aan het einde van een lijn. |
| LineSpacing | SmallInt | TextDoubleSpacing, TextDoubleSpacing2, TextSingleSpacing, TextSingleSpacing2, TextTripleSpacing | Geeft het aantal lege regels dat wordt afgedrukt tussen de tekstregels van een veldobject of een tekstobject. |
| LineStyle | SmallInt | DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine | Geeft de stijl van een regel. |
| LineType | SmallInt | CurvedLine, StraightLine | Geeft het type van een lijn. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| List.Count | SmallInt | Nvt | Geeft het aantal elementen in een lijst. |
| List.Selection | SmallInt | Nvt | Geeft het element dat is geselecteerd in een lijst. |
| Locked | Logical | True, False | Geeft aan of de tabel die is verbonden met een ontwerpobject vergrendeld is. |
| LookupTable | String | Nvt | De naam van de opzoektabel voor een veldobject. |
| LookupType | String | JustCurrentField, AllCorresponding | Geeft het opzoektype. |
| Magnification | SmallInt | Magnify100, Magnify200, Magnify25, Magnify400, Magnify50, MagnifyBestFit | Geeft de weergavevergroting van een bitmap-object. U kunt ook een vaste waarde invoeren. |
| Manager | String | Nvt | Geeft de UIObject-naam van een formulier terug. |
| MarkerPos | LongInt | Nvt | Het "andere einde" van een selectie. Zie ook: CursorPos. |
| MaximizeButton | Logical | True, False | Geeft aan of bepaalt of het venster van een formulier een Vergrootknop heeft. |
| Maximum | String | Nvt | Geeft de maximale waarde die is toegestaan in een veld. |
| MemoView | Logical | True, False | Geeft aan of voor een veldobject de memoweergavemodus is geactiveerd. |
| MinimizeButton | Logical | True, False | Geeft aan of bepaalt of het venster van een formulier een Verkleinknop heeft. |
| Minimum | String | Nvt | Geeft de minimale waarde die is toegestaan in een veld. |
| Modal | Logical | True, False | Geeft aan of een dialoogformulier modaal is. |
| MouseActivate | Logical | True, False | Geeft aan of een dialoogformulier de focus krijgt als gevolg van een MouseEvent. |
| NCols | SmallInt | Nvt | Geeft het aantal kolommen in een tabelframe of multi-record object terug. |
| NRecords | LongInt | Nvt | Geeft het aantal records in de tabel die is verbonden met een ontwerpobject. |
| NRows | SmallInt | Nvt | Geeft het aantal rijen in een tabelframe of multi-record object terug. |
| Name | String | Nvt | Geeft de naam van een ontwerpobject. |
| Next | String | Nvt | Geeft de naam terug van het volgende object in het insluitende object. |
| NoEcho | Logical | True, False | Geeft aan of tekens die worden getypt in een veldobject worden getoond |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| OverStrike | Logical | True, False | Geeft aan of voor een veld- of tekstobject de overschrijfmodus (en niet de invoegmodus) is geactiveerd. |
| Owner | String | Nvt | De naam van het logische insluitende object van een object. |
| Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent | Geeft de kleur van een patroon. |
| Pattern.Style | SmallInt | Bricks, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern | Geeft de stijl van een patroon. |
| PersistView | Logical | True, False | Bepaalt of de veldweergave of de memoweergave geactiveerd moet blijven. |
| Picture | String | Nvt | Een sjabloon die de waarde in een veldobject opmaakt. |
| PinHorizontal | Logical | True, False | Geeft aan of wordt voorkomen dat een object horizontaal wordt verplaatst. |
| PinVertical | Logical | True, False | Geeft aan of wordt voorkomen dat een object verticaal wordt verplaatst. |
| Position | Point | Nvt | Geeft de coördinaten van de rechter bovenhoek van een ontwerpobject. |
| PrecedePageHeader | Logical | True, False | Geeft aan of een rapportzone vóór de paginakopregel moet verschijnen. |
| Prev | String | Nvt | Geeft de naam terug van het vorige object in hetzelfde insluitende object. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| PrintOn1stPage | Logical | True, False | Geeft aan of een rapportzone wordt afgedrukt op de eerste pagina van het rapport. |
| RasterOperation | LongInt | MergePaint, NotSourceCopy, NotSourceErase, SourceAnd, SourceCopy, SourceErase, SourceInvert, SourcePaint | Geeft aan hoe de kleuren in twee overlappende ontwerpobjecten moeten worden gecombineerd. |
| Readonly | Logical | True, False | Geeft aan of een veldobject alleen-lezen is. |
| RecNo | LongInt | Nvt | Geeft de positie van een record. (Bij dBASE-tabellen kan dit veel tijd kosten.) |
| Refresh | Logical | True, False | Geeft aan wanneer gegevens die op het scherm worden weergegeven |
| Required | Logical | True, False | Geeft aan of aan een veldobject een waarde moet zijn toegewezen |
| RowNo | SmallInt | Nvt | Geeft het rijnummer van een record dat wordt weergegeven in een tabelframe, multi-record object of tabelweergave, te beginnen met 1. |
| Scroll | Point | Nvt | Hoe ver u hebt geschoven. |
| SelectedText | String | Nvt | Geeft de geselecteerde tekst in een veldobject terug. |
| SeqNo | LongInt | Nvt | Het werkelijke sequentienummer van een record dat wordt weergegeven, waarbij rekening wordt gehouden met filters en indexen. |
| Shrinkable | Logical | True, False | Geeft aan of een rapportzone kan worden verkleind. |
| Size | Point | Nvt | Geeft de coördinaten van de linker benedenhoek van een ontwerpobject. |
| SizeToFit | Logical | True, False | Als SizeToFit True is, wordt het formulier geopend met de afmeting van de onderliggende pagina. Als SizeToFit False is, wordt het formulier geopend met de afmeting waarmee het was opgeslagen. |
| SortOrder | Logical | True, False | Geeft de sorteervolgorde van een rapport aan.<br>True = Aflopende volgorde,<br>False = Oplopende volgorde. |
| StandardMenu | Logical | True, False | Geeft aan of een formulier de standaardmenu's gebruikt. |
| Start | Point | Nvt | Geeft de coördinaten van het begin van een lijn. Zie ook: End. |
| Style | SmallInt | BorlandButton, WindowsButton | Geeft of bepaalt de weergavestijl van een knop. |

| Kenmerk | Gegevens type | Waarden | Beschrijving |
|---|---|---|---|
| TabStop | Logical | True, False | Geeft aan of een veldobject een tabstop is. |
| TableName | String | Nvt | Geeft de naam van een tabel waarmee een ontwerpobject is verbonden. |
| Text | String | Nvt | Geeft de tekens die worden weergegeven in een tekstobject. |
| ThickFrame | Logical | True, False | Als ThickFrame, DialogForm en Border True zijn, wordt een dik vensterkader gebruikt in plaats van het gebruikelijke kader met een breedte van een pixel. |
| Thickness | SmallInt | LWidth10Points, LWidth1Point, LWidth2Points, LWidth3Points, LWidth6Points, LWidthHairline, LWidthHalfPoint | Geeft de dikte van een lijn. |
| Title | String | Nvt | Geeft de tekst op de titelbalk van een formulier. |
| TopLine | LongInt | Nvt | Het nummer van de regel die op het moment boven in een tekstobject wordt weergegeven. |
| Touched | Logical | True, False | True als de gebruiker wijzigingen heeft aangebracht die nog niet zijn doorgevoerd. |
| Translucent | Logical | True, False | Geeft aan of de kleur van een object doorschijnend is. |
| Value | String | Nvt | Geeft de waarde van een ontwerpobject. |
| VerticalScrollBar | Logical | True, False | Geeft aan of een object een verticale schuifbalk heeft. Niet geldig voor alle UIObjecten. Raadpleeg het overzicht. |
| Visible | Logical | True, False | Geeft aan of een object wordt weergegeven. |
| WordWrap | Logical | True, False | Geeft aan of regelovergangen worden ingevoegd in regels die de breedte van een veldobject overschrijden. |

Table C-3 Specifieke kenmerken voor grafische objecten

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| BackWall.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| BackWall.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| BackWall.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| Background.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Background.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Background.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| BaseFloor.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| BaseFloor.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| BaseFloor.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| BindType | SmallInt | Graph1DSummary, Graph2DSummary, GraphTabular |
| CurrentSeries | SmallInt | Nvt |
| CurrentSlice | SmallInt | Nvt |
| GraphType | SmallInt | Graph2DArea, Graph2DBar, Graph2DColumns, Graph2DLine, Graph2DPie, Graph2DRotatedBar, Graph2DStackedBar, Graph3DArea, Graph3DBar, Graph3DColumns, Graph3DPie, Graph3DRibbon, Graph3DRotatedBar, Graph3DStackedBar, Graph3DStep, Graph3DSurface, Graph3DXY |
| Label.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Label.Font.Size | SmallInt | Afhankelijk van het systeem |
| Label.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| Label.Font.Typeface | String | Afhankelijk van het systeem |
| Label.LabelFormat | SmallInt | GraphHideY, GraphPercent, GraphShowY |
| Label.LabelLocation | SmallInt | LabelAbove, LabelBelow, LabelBottom, labelCenter, LabelLeft, LabelMiddle, LabelRight, LabelTop |
| Label.NumberFormat | Nvt | Opmaakspecificatie |
| LeftWall.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| LeftWall.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| LeftWall.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| LegendBox.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| LegendBox.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| LegendBox.Font.Size | SmallInt | Afhankelijk van het systeem |
| LegendBox.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| LegendBox.Font.Typeface | String | Afhankelijk van het systeem |
| LegendBox.LegendPos | SmallInt | LegendLeft, LegendRight |
| LegendBox.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| LegendBox.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| MaxGroups | SmallInt | Afhankelijk van de grafiek |
| MaxXValues | SmallInt | Afhankelijk van de grafiek |
| MinXValues | SmallInt | Afhankelijk van de grafiek |
| Options.Elevation | SmallInt | 0 tot 90 graden |
| Options.Rotation | SmallInt | 0 tot 90 graden |
| Options.ShowAxes | Logical | True, False |
| Options.ShowGrid | Logical | True, False |
| Options.ShowLabels | Logical | True, False |
| Options.ShowLegend | Logical | True, False |
| Options.ShowTitle | Logical | True, False |
| Series.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Series.Graph_Title.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Series.Graph_Title.Font.Size | SmallInt | Afhankelijk van het systeem |
| Series.Graph_Title.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| Series.Graph_Title.Font.Typeface | String | Afhankelijk van het systeem |
| Series.Graph_Title.Text | String | Nvt |
| Series.Graph_Title.UseDefault | Logical | True, False |
| Series.Line.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| Series.Line.LineStyle | SmallInt | DashDotDotLine, DashDotLine, DashedLine, DottedLine, NoLine, SolidLine |
| Series.Line.Thickness | SmallInt | LWidth10Points, LWidth1Point, LWidth2Points, LWidth3Points, LWidth6Points, LWidthHairline, LWidthHalfPoint |
| Series.Marker | SmallInt | MarkerBoxedCross, MarkerBoxed_Plus, MarkerCross, MarkerFilledBox, MarkerFilledCircle, MarkerFilledDownTriangle, MarkerFilledTriangle, MarkerFilledTriangles, MarkerHollowBox, MarkerHollowCircle, MarkerHollowDownTriangle, MarkerHollowTriangle, MarkerHollowTriangles, MarkerHorizontalLine, MarkerPlus, MarkerVerticalLine |
| Series.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Series.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| Series.TypeOverride | SmallInt | Graph2DArea, Graph2DBar, Graph2DLine, None |
| Slice.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| Slice.Explode | Logical | |
| Slice.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| Slice.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| TitleBox.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| TitleBox.Graph_Title.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| TitleBox.Graph_Title.Font.Size | SmallInt | Afhankelijk van het systeem |
| TitleBox.Graph_Title.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| TitleBox.Graph_Title.Font.Typeface | String | Afhankelijk van het systeem |
| TitleBox.Graph_Title.Text | String | Nvt |
| TitleBox.Graph_Title.UseDefault | Logical | True, False |
| TitleBox.Pattern.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| TitleBox.Pattern.Style | SmallInt | BricksPattern, CrosshatchPattern, DiagonalCrosshatchPattern, DottedLinePattern, EmptyPattern, FuzzyStripesDownPattern, HeavyDotsPattern, HorizontalLinesPattern, LatticePattern, LeftDiagonalLinesPattern, LightDotsPattern, MaximumDotsPattern, MinimumDotsPattern, MediumDotsPattern, RightDiagonalLinesPattern, ScalesPattern, StaggeredLinesPattern, ThickHorizontalLinesPattern, ThickStripesDownPattern, ThickStripesUpPattern, ThickVerticalLinesPattern, VerticalLinesPattern, WeavePattern, ZigZagPattern |
| TitleBox.Subtitle.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| TitleBox.Subtitle.Font.Size | SmallInt | Afhankelijk van het systeem |
| TitleBox.Subtitle.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| TitleBox.Subtitle.Font.Typeface | String | Afhankelijk van het systeem |
| TitleBox.Subtitle.Text | String | Nvt |
| TitleBox.Subtitle.UseDefault | Logical | True, False |
| XAxis.Graph_Title.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| XAxis.Graph_Title.Font.Size | SmallInt | Afhankelijk van het systeem |
| XAxis.Graph_Title.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| XAxis.Graph_Title.Font.Typeface | String | Afhankelijk van het systeem |
| XAxis.Graph_Title.Text | String | Nvt |
| XAxis.Graph_Title.UseDefault | Logical | True, False |
| XAxis.Scale.AutoScale | Logical | True, False |
| XAxis.Scale.HighValue | Number | Afhankelijk van de grafiek |
| XAxis.Scale.Increment | Number | Afhankelijk van de grafiek |
| XAxis.Scale.Logarithmic | Logical | True, False |
| XAxis.Scale.LowValue | Number | Afhankelijk van de grafiek |
| XAxis.Ticks.Alternate | Logical | True, False |
| XAxis.Ticks.DateFormat | Nvt | Opmaakspecificatie |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| XAxis.Ticks.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| XAxis.Ticks.Font.Size | SmallInt | Afhankelijk van het systeem |
| XAxis.Ticks.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| XAxis.Ticks.Font.Typeface | String | Afhankelijk van het systeem |
| XAxis.Ticks.NumberFormat | Nvt | Opmaakspecificatie |
| YAxis.Graph_Title.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| YAxis.Graph_Title.Font.Size | SmallInt | Afhankelijk van het systeem |
| YAxis.Graph_Title.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| YAxis.Graph_Title.Font.Typeface | String | Afhankelijk van het systeem |
| YAxis.Graph_Title.UseDefault | Logical | True, False |
| YAxis.Scale.AutoScale | Logical | True, False |
| YAxis.Scale.HighValue | Number | Afhankelijk van de grafiek |
| YAxis.Scale.Increment | Number | Afhankelijk van de grafiek |
| YAxis.Scale.Logarithmic | Logical | True, False |
| YAxis.Scale.LowValue | Number | Afhankelijk van de grafiek |
| YAxis.Ticks.Alternate | Logical | True, False |
| YAxis.Ticks.DateFormat | Nvt | Opmaakspecificatie |
| YAxis.Ticks.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| YAxis.Ticks.Font.Size | SmallInt | Afhankelijk van het systeem |
| YAxis.Ticks.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| YAxis.Ticks.Font.Typeface | String | Afhankelijk van het systeem |
| YAxis.Ticks.Number.Format | Nvt | Opmaakspecificatie |
| ZAxis.Graph_Title.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| ZAxis.Graph_Title.Font.Size | SmallInt | Afhankelijk van het systeem |

| Kenmerk | Gegevenstype | Waarden |
|---|---|---|
| ZAxis.Graph_Title.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| ZAxis.Graph_Title.Font.Typeface | String | Afhankelijk van het systeem |
| ZAxis.Graph_Title.UseDefault | Logical | True, False |
| ZAxis.Scale.AutoScale | Logical | True, False |
| ZAxis.Scale.HighValue | Number | Afhankelijk van de grafiek |
| ZAxis.Scale.Increment | Number | Afhankelijk van de grafiek |
| ZAxis.Scale.Logarithmic | Logical | True, False |
| ZAxis.Scale.LowValue | Number | Afhankelijk van de grafiek |
| ZAxis.Ticks.Alternate | Logical | True, False |
| ZAxis.Ticks.Font.Color | LongInt | Black, Blue, Brown, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, Gray, Green, LightBlue, Magenta, Red, White, Yellow, Transparent |
| ZAxis.Ticks.Font.Size | SmallInt | Afhankelijk van het systeem |
| ZAxis.Ticks.Font.Style | SmallInt | FontAttribBold, FontAttribItalic, FontAttribNormal, FontAttribStrikeout, FontAttribUnderline |
| ZAxis.Ticks.Font.Typeface | String | Afhankelijk van het systeem |

# Verklarende woordenlijst

**aaneenschakeling** Het samenvoegen van twee of meer reeksen tot één reeks. De operator voor aaneenschakeling is een plus-teken (+).

**aanwijzer** Een visuele markering die de positie van de muis op het scherm aangeeft.

**accolades** De tekens { en }. Accolades markeren commentaar in code.

**actie** De handeling die een methode activeert (die zorgt dat de code wordt uitgevoerd).

**actiegestuurde applicatie** Een applicatie waarin code wordt uitgevoerd in antwoord op acties, in tegenstelling tot een proceduregerichte applicatie, waarin code in een lineaire volgorde wordt uitgevoerd.

**actiemodel** De regels die aangeven hoe acties worden verwerkt door objecten op een formulier.

**active** Een ObjectPAL-variabele die het object vertegenwoordigt dat focus heeft (wordt verderop beschreven).

**afbreekpunt** Een vlag die in broncode wordt gezet om de uitvoering tijdelijk te onderbreken. Afbreekpunten worden gebruikt bij het opsporen van fouten.

**ANSI** Een afkorting van American National Standards Institute; een verzameling acht-bits codes die 256 standaardtekens, letters, getallen en symbolen omvat. De ASCII-tekenset (zie hieronder) komt overeen met de eerste 128 ANSI-tekens.

**applicatie** Een groep formulieren, methodes, queries en procedures die een eenheid vormen, waarin gebruikers gegevens kunnen invoeren, bekijken, onderhouden en weergeven.

**Application** Een ObjectPAL-type dat een handle verschaft voor het bureaublad van Paradox.

**argument** Informatie die wordt doorgegeven aan een methode of een procedure.

**array** Een speciaal objecttype dat uit verschillende afzonderlijke elementen bestaat. Elementen van een array worden aangegeven met subscripten tussen vierkante haken. ar[1] en ar[2] zijn dus de eerste twee elementen van de array met de naam *ar*.

**array-element** Een onderdeel van een array. De array die met de volgende declaratie wordt gemaakt, heeft zeven reekselementen, ar[1] tot en met ar[7].

```
ar Array [7] String
```

**ASCII** Een afkorting van American Standard Code for Information Interchange; een verzameling zeven-bits codes die 128 standaardtekens, letters, getallen en symbolen definiëren. De IBM-PC breidt deze code uit tot 8 bits om bepaalde grafische tekens te kunnen opnemen. Windows-produkten gebruiken de ANSI-tekenset (zie hierboven).

**backslash-code** Een backslash gevolgd door een of meer tekens die een ASCII-teken vertegenwoordigen, bijvoorbeeld \" of \018. Backslash-codes worden gebruikt om aanhalingstekens binnen reeksen te plaatsen en andere tekens op te nemen die een bijzondere betekenis hebben voor Paradox.

**beperkte weergave** Een detailtabel op een multi-tabel formulier, die aan een hoofdtabel is gekoppeld op een één-op-één of één-op-meer basis en alleen de records laat zien die overeenstemmen met het huidige hoofdrecord.

**bereik** De toegankelijkheid of beschikbaarheid van een variabele, methode of procedure voor andere objecten.

**bericht** Een reeksuitdrukking die op de statusregel wordt weergegeven.

**bestand** Een verzameling gegevens die onder één naam zijn opgeslagen op een schijf. Paradox-tabellen worden bijvoorbeeld opgeslagen in bestanden.

**bibliotheek** Een verzameling ObjectPAL-code die kan worden gebruikt door objecten in een of meer formulieren.

**Bureaublad** Het hoofdvenster in Paradox.

**coderen** Een tabel of script vertalen in code die niet kan worden gelezen zonder het juiste wachtwoord.

**constante** Een constante vertegenwoordigt een waarde die niet kan worden veranderd. DataNextRecord is bijvoorbeeld een ObjectPAL-constante die een verplaatsing opgeeft naar het volgende record in een tabel.

**controlestructuur** Een opeenvolging van spronginstructies, zoals **if**...**then**... **endIf** en **while**...**endWhile**, die de volgorde beïnvloedt waarin instructies worden uitgevoerd.

***Ctrl-Break*** Een toetscombinatie die de programma-uitvoering stopt. U kunt Paradox zo configureren dat op ***Ctrl-Break*** wordt geantwoord met het openen van de Debugger.

**Database** Een objecttype dat informatie bevat over relaties tussen tabellen.

**DDE** Afkorting van Dynamic Data Exchange. Een manier waarop Windows-applicaties gegevens gezamenlijk kunnen gebruiken.

**Debugger** Onderdeel van de ObjectPAL-IDE (Integrated Development Environment). Met de Debugger kunt u interactief de uitvoering van opdrachten in uw methodes testen en volgen.

**DLL** Afkorting van Dynamic Link Library. Een DLL is een programma waardoor Windows-programma's code voor de uitvoering van gemeenschappelijke taken gezamenlijk kunnen gebruiken.

**doel** Het object waarvoor een actie is bedoeld. Als u bijvoorbeeld op een knop klikt, is de knop het doel.

**doorvoeren** Veranderingen accepteren in een record en de gegevens in de tabel plaatsen.

**dynamische array** Een speciaal type array waarbij elk element een reeks als index heeft. Bijvoorbeeld: ["Produkt"], ["Paradox voor Windows"], ["Type"] ["Relationele database"], ["Versie"][1.0].

**Editor** Het onderdeel van de Paradox-IDE dat wordt gebruikt om ObjectPAL-methodes te maken en te bewerken.

**Event** Een objecttype.

**FileSystem** Een ObjectPAL-type. FileSystem-variabelen bevatten informatie over schijfbestanden.

**focus** Een attribuut van een object. Een object dat focus heeft (ook wel het actieve object genoemd), kan toetsenbordinvoer behandelen. Het actieve object is meestal gemarkeerd.

**formulier** Een venster dat gegevens en objecten weergeeft. Een formulier is ook een ObjectPAL-type (Form). Het formulier is het hoogste insluitende object.

**functietoetsen** De twaalf toetsen boven aan het toetsenbord. (Sommige toetsenborden hebben tien toetsen links op het toetsenbord, genummerd van *F1* tot en met *F10*.)

**gegevens** De informatie die Paradox opslaat in een tabel.

**gegevenstype** Het type gegevens dat een veld, een variabele of een array-element bevat.

**genormaliseerde gegevensstructuur** Een indeling van gegevens in tabellen waarin elk record het kleinste aantal velden bevat dat nodig is om unieke categorieën te vormen. In plaats van een paar records te gebruiken om alle mogelijke informatie te verschaffen, verspreidt een genormaliseerde tabel gegevens over een groot aantal records, waarbij minder velden worden gebruikt. Een genormaliseerde tabel geeft meer analytische flexibiliteit.

**gereserveerde woorden** De namen van opdrachten, sleutelwoorden, functies, systeemvariabelen en operatoren. Deze woorden mogen niet worden gebruikt als ObjectPAL-variabelen of array-namen. *Zie ook* sleutelwoord.

**globale variabele** Een variabele die beschikbaar is voor alle objecten op een formulier. *Zie ook* lokale variabele.

**handle** Een variabele die u in code kunt gebruiken om objecten te manipuleren.

**Help** Het online Helpsysteem van Paradox. U kunt in Paradox op elk moment op *F1* drukken om informatie te krijgen over de huidige bewerking.

**hiërarchie** De onderlinge relatie van objecten op een formulier, afgeleid van hun visuele en ruimtelijke relatie. *Zie ook* insluiting

**impasse** Een situatie die in een multi-user-omgeving ontstaat als tegelijkertijd twee vergrendelingsopdrachten worden gegeven die niet compatibel zijn.

**index** Een bestand dat de volgorde bepaalt waarin Paradox toegang kan krijgen tot de records in een tabel. Het sleutelveld van een Paradox-tabel legt de primaire index van de tabel vast. *Zie ook* sleutel, secundaire index.

**ingebouwde methode** Voorgedefinieerde code die wordt gekoppeld aan elk object dat u op een formulier kunt plaatsen. Ingebouwde methodes definiëren het standaardantwoord van een object op acties.

**insluiting** Een object sluit een ander object in als het laatste object zich volledig binnen de grenzen van het eerste object bevindt. Insluitrelaties hebben invloed op de beschikbaarheid van variabelen, methodes en procedures.

**inspecteren** De kenmerken van een object bekijken of veranderen. Als u een object wilt inspecteren, klikt u rechts op het object of selecteert u het met het toetsenbord door op *F6* te drukken. Het menu van het object verschijnt dan. Kies in het menu het kenmerk dat u wilt veranderen.

**invoegpositie** De plaats waar tekst wordt ingevoegd als u typt. De invoegpositie is meestal een knipperend verticaal streepje.

**kenmerken** De attributen van een object. U klikt rechts op een object om de kenmerken te bekijken of te veranderen. *Zie ook* inspecteren.

**kolom** Een verticaal onderdeel van een Paradox-tabel dat één veld bevat. In een Paradox-rapport is een kolom een verticaal gebied met een of meer velden.

**koppelsleutel** In een gekoppeld multi-tabel formulier is een koppelsleutel het deel van de sleutel van de ondergeschikte tabel dat is gekoppeld aan of overeenstemt met velden in de hoofdtabel.

**leeg** Een veld dat of een variabele die geen waarde bevat.

**levensduur** De periode waarin een element actief of beschikbaar is.

**logische operator** Een van de drie operatoren (AND, OR en NOT) die kunnen worden gebruikt voor logische gegevens. Een AND-operator tussen twee waarden geeft bijvoorbeeld een logische True-waarde als resultaat als beide oorspronkelijke waarden ook True zijn. Deze operatoren worden ook wel Booleaanse operatoren genoemd.

**logische waarde** Een waarde (True of False) die wordt toegewezen aan een uitdrukking als deze wordt geëvalueerd. Wordt ook een Booleaanse waarde genoemd.

**lokale variabele** Een variabele die alleen beschikbaar is voor de methode of procedure waarin deze wordt gedeclareerd. *Zie ook* globale variabele.

**lussen** Controlestructuren die een serie opdrachten herhalen totdat aan een bepaalde voorwaarde is voldaan. *Zie ook* controlestructuren.

**menu** Een weergave van de keuzes of opties die beschikbaar zijn. Als u ObjectPAL gebruikt, kunt u zowel applicatiemenu's als pop-up menu's maken en bewerken.

**menukeuze** Een opdracht die u in een menu kiest.

**methode** ObjectPAL-code die is gekoppeld aan een object en die het antwoord van het object op een actie definieert.

**object** Een geheel van code en gegevens. Alle eenheden die in Paradox kunnen worden gemanipuleerd, zijn objecten.

**Objectenschema** Een diagram dat laat zien hoe objecten op een formulier zijn verbonden door insluitrelaties.

**omhoogborrelen** Een proces waarbij acties via het doelobject naar boven worden doorgegeven in de hiërarchie van ingesloten objecten.

**opmaakspecificatie** De manier waarop een veldwaarde op het scherm wordt getoond of wordt uitgevoerd naar een printer.

**parameter** De variabele waarin een argument wordt doorgegeven. Parameters worden gebruikt bij de definitie van procedures.

**pixel** Eén punt op het scherm. Het woord is een samentrekking van *pic*ture (beeld) en *el*ement

**primaire index** Een index op de sleutelvelden van een Paradox-tabel. Een primaire index bepaalt de lokatie van records, biedt u de mogelijkheid de tabel als detailtabel te gebruiken in een koppeling, houdt records in de sorteervolgorde en versnelt bewerkingen. *Zie ook* secundaire index.

**procedure** Code die is geplaatst tussen de sleutelwoorden PROC en ENDPROC. In tegenstelling tot een methode krijgt een procedure geen context van een object.

**prompt** Instructies die op het scherm, meestal op de statusbalk, verschijnen. Een prompt vraagt om informatie of begeleidt de gebruiker tijdens een bewerking.

**punt** Een geordend getallenpaar dat een lokatie vertegenwoordigt op het scherm.

**QBE** *Zie* query by example.

**query** Een vraag die u stelt over gegevens in een Paradox-tabel en die u formuleert op een query-formulier. Query is ook een ObjectPAL-type.

**query by example (QBE)** De methode om vragen over gegevens te stellen door voorbeelden te geven van de antwoorden die u zoekt.

**rasterbewerking** Een bewerking die aangeeft hoe kleuren op het scherm verschijnen.

**record** Een horizontale rij in een Paradox-tabel die een groep verbonden gegevensvelden bevat. Record is ook een ObjectPAL-type.

**recordnummer** Een uniek nummer dat elk record in een Paradox-tabel identificeert.

**reeks** Een alfanumerieke waarde of een uitdrukking die bestaat uit alfanumerieke tekens.

**reeks tussen aanhalingstekens** Tekst tussen dubbele aanhalingstekens.

**relationele database** Een database-ontwerp volgens principes die het *relationele model* worden genoemd. Gegevens in een relationele database worden in tabellen ingedeeld.

**rij** Een horizontaal onderdeel van een tabel, dat in Paradox een record word genoemd.

**runtime bibliotheek** Een verzameling voorgedefinieerde methodes en procedures die werken op objecten van bepaalde types.

**runtime fout** Een fout die ontstaat als een syntactisch geldige instructie niet kan worden uitgevoerd in de huidige context.

**samengesteld object** Een object dat is samengesteld uit twee of meer objecten. Een tabelframe is bijvoorbeeld een samengesteld object dat bestaat uit veldobjecten en recordobjecten.

**script** Een verzameling ObjectPAL-code die wordt uitgevoerd zonder dat er een venster wordt geopend.

**secundaire index** Een index die wordt gebruikt voor koppelingen, queries en het veranderen van de weergavevolgorde van tabellen.

**Self** Een ObjectPAL-variabele. Self verwijst naar het object waaraan de code die wordt uitgevoerd, is gekoppeld.

**sessie** Een kanaal naar de database-engine.

**Session** Een ObjectPAL-type.

**sjabloon** Een patroon van tekens dat bepaalt wat een gebruiker, in antwoord op een prompt, in een veld kan typen tijdens bewerking of gegevensinvoer.

**sleutel** Een of meer velden die worden gebruikt om records te rangschikken. *Zie ook* sleutelveld.

**sleutelveld** Een veld dat is aangewezen als (deel van een) aanduiding van de records in een Paradox-tabel. Een sleutel heeft drie effecten: er wordt voorkomen dat de tabel dubbele records bevat, de records worden op basis van de sleutelvelden op gesorteerde volgorde bijgehouden en er wordt een primaire index gemaakt voor de tabel. *Zie ook* index.

**sleutelwoord** Een woord dat is gereserveerd door ObjectPAL. U kunt een sleutelwoord niet gebruiken als de naam van een variabele, array, methode of procedure.

**sprongopdrachten** Controlestructuren die opties uitvoeren, afhankelijk van de vraag of er aan bepaalde voorwaarden wordt voldaan. Voorbeelden: **if** en **while**.

**String** Een ObjectPAL-type.

**structuur** De rangschikking van velden in een tabel.

**subject** Het object waarmee een eigen methode wordt aangeroepen. In de volgende instructie is bijvoorbeeld *hetKader* het subject.

```
hetKader.doeIets()
```

**subreeks** Elk onderdeel van een reeks.

**syntaxisfout** Een fout die te wijten is aan een verkeerd uitgedrukte instructie.

**tabelweergave** De weergave van een tabel in rijen en kolommen in de Paradox-tabelopmaak.

**TableView** Een ObjectPAL-type.

**TCursor** Een ObjectPAL-type: een verwijzing naar gegevens in een tabel. Met TCursors kunt u gegevens manipuleren, zonder dat u de huidige tabel hoeft weer te geven.

**tilde-variabele** Een variabele die in een query-formulier wordt gebruikt, en die moet worden voorafgegaan door een tilde (~).

**stapsgewijze ontwikkeling** Een vorm van applicatie-ontwikkeling waarbij kleine gedeelten of de algemene structuur van de applicatie interactief worden ontworpen en getest.

**toetscode** Een code die een toetsenbordteken vertegenwoordigt in ObjectPAL-methodes. Deze code kan een ANSI-getal zijn of een reeks die een toetsnaam vertegenwoordigt en die bekend is bij Paradox.

**transactie** Een groep verbonden wijzigingen in een database.

**twip** Een maateenheid die gelijk is aan 1/1440 van een inch (1/20 van een printerpunt).

**type** Een manier om objecten met dezelfde attributen te classificeren. Alle tabellen en alle formulieren hebben bijvoorbeeld attributen gemeen, maar tabellen en formulieren hebben andere attributen. Tabellen en formulieren behoren dus tot verschillende types.

**uitdrukking** Een groep tekens die gegevenswaarden, variabelen, arrays, operatoren of functies kan bevatten die een hoeveelheid of een waarde vertegenwoordigen. Een uitdrukking kan een bepaald gegevenstype opleveren of, in bepaalde gevallen, eerst worden geconverteerd naar reekswaarden voordat de waarde-uitdrukking wordt geëvalueerd.

**uitgebreide IBM-codes** Toetsen of toetscombinaties op het toetsenbord die niet overeenkomen met de standaard ASCII-tekencodes en die speciale "uitgebreide codenummers" tussen -1 en -132 hebben gekregen.

**validiteitscontrole** Een beperking op de waarden die u in een veld kunt invoeren.

**variabele** Een plaats in het geheugen om gegevens tijdelijk op te slaan.

**veld** Een gegevensonderdeel in een tabel. Een verzameling van verbonden velden is een record.

**veldobject** Een UIObject dat wel of niet kan worden geassocieerd met een veld in een tabel.

**veldtoewijzing** Het gebruik van puntnotatie om de waarde van een uitdrukking toe te wijzen aan een veld.

**veldtype** Het type informatie dat in een veld van een tabel kan worden ingevoerd. *Zie ook* Gegevenstype.

**veldwaarde** De gegevens uit een veld van een record. Als er geen gegevens zijn, is het veld leeg. Veldobjecten beschikken over een kenmerk 'Value'.

**Veldweergave** In veldweergave kunt u de invoegpositie teken voor teken door een veld verplaatsen. Veldweergave wordt gebruikt om veldwaarden te bekijken die te groot zijn om in de huidige veldbreedte te worden weergegeven, of om een veldwaarde te bewerken.

**voorbeeldelement** In een query-opdracht: een willekeurige tekenreeks die elke waarde in een veld vertegenwoordigt. In Paradox geeft u een voorbeeldelement op door op Voorbeeld *F5* te drukken en de tekens in het query-beeld te typen. In methodes geeft u voorbeeldelementen op door een onderstrepingsteken voor de tekens te plaatsen.

**Weergavebeheer** Een categorie objecttypes die de types 'Application', 'Form', 'Report' en 'Table View' bevat.

## A

## B

## C

## D

## E

## F

## G

## H

## I

## J



## K

## L



## M

## N



## O

## P

## Q



## R

## S

## T

## U



## V

## W

## X



## Y



## Z

**Borland**

Borland, dat haar hoofdkantoor heeft in Amerika, heeft verder kantoren in Australië, België, Canada, Denemarken, Duitsland, Engeland, Frankrijk, Hong Kong, Italië, Japan, Korea, Maleisië, Nederland, Nieuw-Zeeland, Singapore, Spanje, Taiwan en Zweden. Nederland: Borland Benelux B.V., De Cuserstraat 93, 1081 CN Amsterdam. België: Borland Belgium N.V., Boechoutlaan 55, Bus 1, 1853 Strombeek-Bever. ■ Part # PDX1145NL2177D ■