Keywords SDWrite

versie 3.0, 1993

Inhoudsopgave

Voorwoord

SDWrite is de speciaal voor SDW ontwikkelde report-writer, die de gebruiker van SDW in staat stelt op eenvoudige wijze en naar eigen inzicht overzichten van de Systeem Encyclopedie samen te stellen en vorm te geven.

Er kan op twee manieren met SDWrite gewerkt worden:

- online
- met behulp van scripts

Online werken met SDWrite

Het is mogelijk SDWrite rechtstreeks via het toetsenbord opdrachten te geven. Dit wordt online werken genoemd. Het online werken met SDWrite is eigenlijk uitsluitend zinvol voor korte, eenmalige rapportage-opdrachten.

Indien SDWrite vanuit een module wordt gestart, is online werken **niet** mogelijk. In dat geval wordt altijd gewerkt met scripts.

Scripts

Het is ook mogelijk SDWrite-opdrachten in een ASCII-file op te nemen en deze file door SDWrite te laten verwerken. Zo'n invoerfile wordt een script genoemd.
SDWrite werkt interpretatief, zodat de scripts niet eerst gecompileerd hoeven te worden voordat ze aan SDWrite worden aangeboden. Dit maakt testen van een script tot een eenvoudige bezigheid: het maken van wijzigingen in het script is voldoende.

Scripts kunnen naar keuze bewaard worden in de directory van het betreffende SDW-systeem, de LIBWS-directory, de LIB-directory of de SCRIPT-directory.
Standaard heeft een script de extensie '.SCR'.

Het werken met scripts heeft in vrijwel alle gevallen de voorkeur, omdat de

script-files eenvoudig aangepast kunnen worden en bovendien herhaalde malen kunnen worden gebruikt. Online commando's dienen elke keer opnieuw te worden ingetoetst.

De commando-syntax is in beide gevallen identiek.

Uitgebreide beschrijving SDWrite
SDWorkstation 2: SDWrite

Alfabetisch overzicht SDWrite-keywords

In dit deel van de handleiding worden op alfabetische volgorde alle keywords besproken die gebruikt kunnen worden in SDWrite.

De beschrijving van elk keyword bestaat uit de volgende onderdelen:

- de syntax van het keyword
- een korte aanduiding van de functie van het keyword
- een uitgebreide beschrijving van de mogelijkheden en beperkingen van het keyword
- een verwijzing naar verwante keywords

Wat deze onderdelen inhouden, komt hierna aan de orde.

Syntax

Aangegeven wordt welke parameters bij het keyword horen en van welke aard deze parameters dienen te zijn.
Daarbij worden de volgende regels gevolgd:

< x >	=	parameter x is verplicht.
< x | y >	=	parameter verplicht, keuze uit parameter x of y.
[x]	=	parameter x is optioneel.
[x | y]	=	parameter optioneel, keuze uit parameter x of y.
[x [y]]	=	beide parameters optioneel, parameter y kan echter alleen voorkomen als ook parameter x voorkomt en is dan niet verplicht.
{ x }	=	herhaling, d.w.z. 0 (nul) of meer malen de parameter x.
{ x | y }	=	herhaling, d.w.z. 0 (nul) of meer malen een van de parameters x en y of een combinatie van beide.

Indien de parameter-aanduiding wordt voorafgegaan door $, dan dient op deze plaats de naam van een variabele te worden opgenomen. Indien het $-teken ontbreekt, dient een niet-variabele parameter te worden meegegeven.

Voorbeeld:

```
add  < x | $x >  < y | $y >
```

betekent dat het keyword **ADD** twee parameters heeft, die beide zowel een variabele als ene niet-variabele kunnen zijn.

Functie

Korte aanduiding van de functie van het betreffende keyword, met tussen haakjes een verwijzing naar het type keyword. Er wordt een onderscheid gemaakt tussen de volgende typen SDWrite-keywords:

- SDW — deze keywords zijn alleen van belang in scripts die een SDW-systeem betreffen
- Lay Out — deze keywords zijn van belang voor de vormgeving van een rapport
- VAR — deze keywords zijn van belang bij het werken met variabelen
- PRG — deze keywords zijn van belang bij het maken (programmeren) van SDWrite-scripts
- IO — deze keywords hebben, in ruime zin, te maken met invoer en uitvoer
- \- — deze keywords zijn van een ander type dan de hiervoor beschrevene

Voorbeeld:

(SELECT_ITEM)

Functie: selecteert van het current component-type de rubriek met de opgegeven naam. (SDW)

Werking

Uitgebreide beschrijving van de mogelijkheden van het betreffende keyword, met informatie over mogelijke toepassingen. Eventueel worden een of meer voorbeelden gegeven. Ook wordt aangegeven welke vereisten en beperkingen

bij het betreffende keyword gelden.

Zie ook

Verwijzing naar de belangrijkste verwante SDWrite-keywords, d.w.z. keywords die een soortgelijke functie hebben als het betreffende keyword of die bij het gebruik van het keyword een belangrijke rol spelen.

ADD

Syntax: ADD <$varnaam> <waarde | $waarde>

Functie: telt het tweede argument op bij het eerste. (VAR)

Werking: Met **ADD** wordt de rekenkundige bewerking optellen uitgevoerd. Daarbij wordt uitgegaan van integers, d.w.z. gehele getallen (...,-2, -1, 0, 1, 2, ...).
De variabele dient vooraf met het **SET**-commando te zijn geïnitialiseerd. Is dat niet gebeurd, dan volgt een foutmelding en wordt de verwerking van het script afgebroken.
Indien de variabele een waarde heeft of krijgt die geen integer is (bijvoorbeeld een woord of een getal met decimalen), dan wordt het integer-gedeelte van de betreffende variabele gebruikt. Dat wil zeggen dat het gedeelte achter de komma wordt genegeerd. Er vindt derhalve **geen** afronding plaats.
Hetzelfde gebeurt met de waarde die als tweede parameter wordt meegegeven.
De opdrachten:

```
add $x 3        (x = computer)
add $x 3        (x = 2,1)
add $x 3,5      (x = 2,1)
add $x 3,5      (x = 2)
```

hebben derhalve als resultaat:

```
x = 3           (0 + 3)
x = 5           (2 + 3)
x = 5           (2 + 3)
x = 5           (2 + 3)
```

Het is ook mogelijk als tweede waarde een variabele mee te geven. Het eerste argument **moet** een variabele zijn. Is dat niet zo, dan wordt de verwerking van het script afgebroken.

Zie ook: **DIVIDE, MULTIPLY, SET, SUBTRACT.**

ASCII

Syntax: ASCII <waarde | $varnaam>

Functie: stuurt de opgegeven of berekende ASCII-waarde naar het uitvoer-device. (Lay Out)

Werking: Met behulp van het keyword **ASCII** kan de gebruiker vanuit een SDWrite-script ASCII-codes naar het uitvoer-device sturen. Dit is vooral handig wanneer de uitvoer naar de printer gaat. Met behulp van ASCII-codes kan de printer namelijk in een bepaalde mode (bijvoorbeeld condensed) gezet worden.
Welke ASCII-code daarvoor gebruikt moet worden is afhankelijk van de gebruikte printer. Zie daarvoor de handleiding die bij de printer hoort.
Natuurlijk is het ook mogelijk om met **ASCII** een letterteken naar het uitvoer-device te sturen. Daarbij wordt dan **geen** newline gegeven, ook niet indien **NEWLINE** ON staat.
Derhalve levert de opeenvolging van:

```
ascii 65
ascii 66
ascii 67
```

als uitvoer op:

```
ABC
```

zònder newline.

ASCII kan tevens gebruikt worden om de ASCII-waarde van een karakter te bepalen. Als een argument wordt meegegeven dat geen getal is, wordt de ASCII-waarde van dat argument geschreven:

```
ascii A
```

levert derhalve als uitvoer:

```
65
```

Zie ook: **NEWLINE.**

BEGIN

Syntax: BEGIN

Functie: duidt het begin aan van een serie bij elkaar horende commando's. (PRG)

Werking: Het keyword **BEGIN** duidt op zichzelf geen handeling aan. Het dient altijd gebruikt te worden in combinatie met **END** en een aantal andere statements die tussen **BEGIN** en **END** in moeten staan.
De aanduiding **BEGIN** en **END** om een aantal statements betekent, dat die statements bij elkaar horen.
Dit is vooral van belang in constructies met **IF** en **WHILE**, maar ook bij het maken van een **HEADER** en een **MACRO**.
Vergelijk de volgende twee script-gedeeltes:

(A.)
```
if equal $var 12
    print regel1
    print regel2
```

(B.)
```
if equal $var 12
    BEGIN
    print regel1
    print regel2
    END
```

Ervan uitgaande dat $var inderdaad gelijk is aan 12 zullen beide scripts dezelfde uitvoer opleveren. Script A print regel1 omdat aan de voorwaarde in het **IF**-statement wordt voldaan, en regel2 omdat het tweede print-statement niet met de **IF**-constructie verbonden is en derhalve altijd geprint wordt.
Script B print zowel regel1 als regel2 omdat aan de voorwaarde in het **IF**-statement wordt voldaan.

Indien $var **niet** gelijk is aan 12, leveren de scripts echter niet dezelfde uitvoer.
Script A zal regel1 niet printen, omdat niet aan de voorwaarde wordt voldaan. Maar regel2 wordt door script A wèl geprint,

omdat dat een gewoon print-statement is.
In script B zijn echter beide print-statements verbonden aan de **IF**-constructie, omdat ze tussen **BEGIN** en **END** staan. Script B levert in dit geval derhalve géén uitvoer.

Overigens zou het verschil tussen beide scripts door een iets andere lay out van script A al veel duidelijker zijn uitgekomen. Vergelijk:

```
(A'.)
if equal $var 12
    print regel1
print regel2
```

Door alleen de bij de **IF**-constructie behorende print-opdracht te laten inspringen, wordt duidelijk dat de tweede print-opdracht van een andere aard is.
Dit is overigens niets meer dan een goede lay out-gewoonte: voor de verwerking van een script doet het al dan niet inspringen niet terzake. De SDWrite-interpreter zal inspringende regels niet anders vertalen dan niet inspringende!

In combinatie met **MACRO** en **HEADER** zijn **BEGIN** en **END** beslissend voor welke opdrachten als onderdeel van de macro respectievelijk header worden gezien.
De opbouw van een macro-definitie en een header-definitie zijn hetzelfde:

```
HEADER
    <definitie>
```

en

```
MACRO <naam>
    <definitie>
```

waarbij de definitie kan bestaan uit:

```
BEGIN
<opdracht(en)>
END
```

of:

<opdracht> (maximaal 1)

Indien de definitie van de macro of header uit meerdere opdrachten bestaat, dan **moet** gebruik gemaakt worden van **BEGIN** en **END**.
Wordt dat niet gedaan, dan zal alleen de eerste opdracht onder 'HEADER' of 'MACRO <naam>' als definitie beschouwd worden.
BEGIN en **END** werken hier dus hetzelfde als in het voorbeeld hiervoor.

Zie ook: **END, IF, HEADER, MACRO, WHILE.**

CLEAR_SCREEN

Syntax: CLEAR_SCREEN

Functie: leegmaken van het scherm. (IO)

Werking: Met behulp van het keyword **CLEAR_SCREEN** kan de gebruiker vanuit een SDWrite-script het scherm leeg maken. Hiermee wordt ervoor gezorgd dat de scherm-uitvoer van het script op een 'nieuwe' scherm-pagina begint.

Zie ook: -.

COLLECTION_ADD

Syntax: COLLECTION_ADD <collectie|$col> <element|$elm>

Functie: toevoegen van een element aan een collectie. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend).
Vervolgens kunnen nieuwe elementen aan de collectie worden toegevoegd met **COLLECTION_ADD**.
Om bijvoorbeeld het element X toe te voegen aan collectie Y, dient de volgende opdracht te worden gegeven:

```
collection_add Y X
```

Het is ook mogelijk de collectie- en element-naam in de vorm van variabelen aan te bieden, bijvoorbeeld:

```
set $collect regel1
set $collection CollectieA
collection_add $collection $collect
```

De volgende keer dat een **COLLECTION_ADD**-commando wordt gegeven voor dezelfde collection, wordt een volgend element aan die collectie toegevoegd.

Zie ook: **COLLECTION_NEW, COLLECTION_NEXT, COLLECTION_PRINT, COLLECTION_FIND, COLLECTION_RESET, COLLECTION_CLOSE, COLLECTION_DELETE.**

COLLECTION_CLOSE

Syntax: COLLECTION_CLOSE <collectie | $collectie>

Functie: afsluiten en verwijderen van een collectie. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend). Vervolgens kunnen op de betreffende collectie alle reguliere handelingen worden uitgevoerd.
Indien de collectie niet langer nodig is, kan deze gesloten en verwijderd worden door **COLLECTION_CLOSE**. Alle gegevens in de collectie gaan hierbij verloren, zodat bij gewenst hergebruik de collectie opnieuw geopend en gevuld zal moeten worden.

Zie ook: **COLLECTION_NEW**, **COLLECTION_NEXT**, **COLLECTION_PRINT**, **COLLECTION_FIND**, **COLLECTION_RESET**, **COLLECTION_ADD**, **COLLECTION_DELETE**.

COLLECTION_DELETE

Syntax: COLLECTION_DELETE <collectie | $collectie>

Functie: verwijdert het geselecteerde element uit de collectie. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend).
Om later handelingen op de elementen te kunnen uitvoeren, dient het gewenste element eerst geselecteerd te worden. Hiervoor dient het keyword **COLLECTION_NEXT**. Ook kan naar een specifiek element 'gesprongen' worden met **COLLECTION_FIND**.

Het selecteren van collectie-elementen is als volgt voor te stellen:
Indien een element in een collection wordt geselecteerd, dan wordt bij dat element als het ware een markering geplaatst. Dankzij die markering weet SDWrite op welk element een commando als **COLLECTION_DELETE** betrekking heeft.
Het commando **COLLECTION_NEXT** zorgt ervoor dat de markering één naam verder wordt geplaatst.
Indien er nog geen element is geselecteerd, wordt door **COLLECTION_NEXT** het eerste element van de betreffende collectie geselecteerd.
Het commando **COLLECTION_FIND** zorgt ervoor dat de markering bij de aangegeven naam wordt geplaatst.

Het commando **COLLECTION_DELETE** zorgt ervoor dat de geselecteerde naam (de naam met de markering) wordt verwijderd uit de collectie.

Het volgende script zoekt en verwijdert het element 'weg' in de collectie 'A':

```
if collection_find A weg
    collection_delete A
```

Zie ook: **COLLECTION_NEW, COLLECTION_NEXT, COLLECTION_PRINT, COLLECTION_ADD, COLLECTION_FIND, COLLECTION_RESET, COLLECTION_CLOSE.**

COLLECTION_FIND

Syntax: COLLECTION_FIND <collectie|$col> <element|$elm>

Functie: zoeken en selecteren van een element in een collectie. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend).
Om handelingen op een element te kunnen uitvoeren, dient het gewenste element eerst geselecteerd te worden. Hiervoor kan naast het keyword **COLLECTION_NEXT** ook **COLLECTION_FIND** gebruikt worden.

Het selecteren van collectie-elementen is als volgt voor te stellen:
Indien een element in een collection wordt geselecteerd, dan wordt bij dat element als het ware een markering geplaatst. Dankzij die markering weet SDWrite op welk element een commando als **COLLECTION_PRINT** betrekking heeft. Het commando **COLLECTION_NEXT** zorgt ervoor dat de markering één naam verder wordt geplaatst.Indien er nog geen element is geselecteerd, wordt door het commando **COLLECTION_NEXT** het eerste element van de betreffende collectie geselecteerd.
Het commando **COLLECTION_FIND** zorgt ervoor dat de markering direct bij de aangegeven naam wordt geplaatst.

Indien in een collectie A de elementen 1, 2 en 3 voorkomen, kan element 2 derhalve met **COLLECTION_NEXT** geselecteerd worden:

```
collection_next A     (* selectie element 1 *)
collection_next A     (* selectie element 2 *)
```

maar ook (rechtstreeks) met **COLLECTION_FIND**:

```
collection_find A 2
```

Het is ook mogelijk om **COLLECTION_FIND** op te nemen in een voorwaardelijke constructie, bijvoorbeeld met **IF**:

```
if collection_find A element1
        collection_print A
```

In dit geval zal element1 geschreven worden naar het current uitvoerkanaal als het aanwezig is in collectie A.
Het is belangrijk op te merken dat **COLLECTION_FIND** ook in dit geval het element niet alleen vindt maar ook **selecteert**. Het resultaat van de **IF**-opdracht is derhalve niet alleen een ja of nee (gevonden of niet gevonden), maar eventueel ook een verschuiving van de markering binnen de collectie.

Welke gevolgen dit kan hebben, toont het volgende voorbeeld:

In de collectie A zijn een viertal elementen opgenomen:

```
aaa
bbb
ccc
ddd
```

Indien het element 'ccc' in de collectie aanwezig is, dienen alle elementen uit de collectie geprint te worden.

Het volgende script doet dit **niet**:

```
if collection_find A ccc
    BEGIN
    while collection_next A
        collection_print A
    END
```

Het resultaat van dit script is:

```
ddd
```

Het element 'ccc' wordt namelijk **zowel gevonden als geselecteerd**, waardoor de opdracht 'collection_next A' de markering bij 'ddd' zal plaatsen.

Om het gewenste resultaat te verkrijgen, moet het script er daarom als volgt uitzien:

```
if collection_find A ccc
    BEGIN
    collection_reset A
    while collection_next A
        collection_print A
    END
```

Zie ook: **COLLECTION_NEW, COLLECTION_NEXT, COLLECTION_PRINT, COLLECTION_ADD, COLLECTION_RESET, COLLECTION_CLOSE, COLLECTION_DELETE.**

COLLECTION_NEW

Syntax: COLLECTION_NEW <collectie | $collectie>

Functie: openen van een nieuwe collectie. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend). Vervolgens kunnen op de betreffende collectie alle reguliere handelingen worden uitgevoerd.
Indien de collectie niet langer nodig is, kan deze gesloten en verwijderd worden door **COLLECTION_CLOSE**. Alle gegevens in de collectie gaan hierbij verloren, zodat bij gewenst hergebruik de collectie opnieuw geopend en gevuld zal moeten worden.

Het is **niet** mogelijk om een reeds geopende collectie nogmaals te openen. Het commando **COLLECTION_NEW** mag derhalve uitsluitend gebruikt worden voor nieuwe collecties of voor collecties die met **COLLECTION_CLOSE** gesloten zijn.

Zie ook: **COLLECTION_FIND, COLLECTION_NEXT, COLLECTION_PRINT, COLLECTION_ADD, COLLECTION_RESET, COLLECTION_CLOSE, COLLECTION_DELETE.**

COLLECTION_NEXT

Syntax: COLLECTION_NEXT <collection | $collection>

Functie: volgende element in een collectie selecteren. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend).
Vervolgens kunnen nieuwe elementen aan de collectie worden toegevoegd met **COLLECTION_ADD**. Om later handelingen op de te bewerken elementen te kunnen uitvoeren, dient het gewenste element eerst geselecteerd te worden. Hiervoor kunnen **COLLECTION_NEXT** en **COLLECTION_FIND** gebruikt worden.

Het selecteren van collectie-elementen is als volgt voor te stellen:
Indien een element in een collection wordt geselecteerd, dan wordt bij dat element als het ware een markering geplaatst. Dankzij die markering weet SDWrite op welk element een commando als **COLLECTION_PRINT** betrekking heeft.
Het commando **COLLECTION_NEXT** zorgt ervoor dat de markering één naam verder wordt geplaatst.
Indien er nog geen element is geselecteerd, wordt door **COLLECTION_NEXT** het eerste element van de betreffende collectie geselecteerd.

Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van elementen in de geselecteerde collectie worden afgewerkt.

Het volgende script schrijft bijvoorbeeld alle elementen in de collectie A:

```
while collection_next A   (* a *)
```

```
collection_print A          (* b *)
```

Zolang er nog een volgende element in collectie A is (a), wordt dit element geprint (b). Is er geen volgend element meer (a), dan eindigt het script.

Zie ook: **COLLECTION_FIND, COLLECTION_NEW, COLLECTION_PRINT, COLLECTION_ADD, COLLECTION_RESET, COLLECTION_CLOSE, COLLECTION_DELETE.**

COLLECTION_PRINT

Syntax: COLLECTION_PRINT <collectie | $collectie>

Functie: schrijft het geselecteerde element van de collectie naar het current uitvoerkanaal of een variabele. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend).
Om handelingen op de elementen te kunnen uitvoeren, dient het gewenste element eerst geselecteerd te worden. Hiervoor kunnen **COLLECTION_NEXT** en **COLLECTION_FIND** gebruikt worden.

Het selecteren van collectie-elementen is als volgt voor te stellen:
Indien een element in een collection wordt geselecteerd, dan wordt bij dat element als het ware een markering geplaatst. Dankzij die markering weet SDWrite op welk element een commando als **COLLECTION_PRINT** betrekking heeft.
Het commando **COLLECTION_NEXT** zorgt ervoor dat de markering één naam verder wordt geplaatst.
Indien er nog geen element is geselecteerd, wordt door **COLLECTION_NEXT** het eerste element van de betreffende collectie geselecteerd.

Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie kan de hele lijst van elementen in de geselecteerde collectie worden afgewerkt. Door in de lus het **COLLECTION_PRINT**-commando op te nemen, kunnen derhalve alle elementen uit een collectie geprint worden.

Het volgende script schrijft bijvoorbeeld alle elementen in de collectie A:

```
while collection_next A        (* a *)
    BEGIN
    collection_print A         (* b *)
    END
```

Zolang er nog een volgende element in collectie A is (a), wordt dit element geprint (b). Is er geen volgend element meer (a), dan eindigt het script.

Zie ook: **COLLECTION_FIND, COLLECTION_NEW, COLLECTION_NEXT, COLLECTION_ADD, COLLECTION_RESET, COLLECTION_CLOSE, COLLECTION_DELETE.**

COLLECTION_RESET

Syntax: COLLECTION_RESET <collectie | $collectie>

Functie: resetten van de selectie-markering in een collectie. (IO)

Werking: Collecties zijn tijdelijke verzamelingen die in het geheugen worden bewerkt en geraadpleegd.

Om gegevens in een collectie te kunnen benaderen, dient deze collectie eerst geopend te worden. Dit kan met het commando **COLLECTION_NEW** (waarmee een specifieke collectie wordt geopend).
Om handelingen op de elementen te kunnen uitvoeren, dient het gewenste element eerst geselecteerd te worden. Hiervoor kunnen **COLLECTION_NEXT** en **COLLECTION_FIND** gebruikt worden.

Het selecteren van collectie-elementen is als volgt voor te stellen:
Indien een element in een collection wordt geselecteerd, dan wordt bij dat element als het ware een markering geplaatst. Dankzij die markering weet SDWrite op welk element een commando als **COLLECTION_PRINT** betrekking heeft.
Het commando **COLLECTION_NEXT** zorgt ervoor dat de markering één naam verder wordt geplaatst.
Indien er nog geen element is geselecteerd, wordt door **COLLECTION_NEXT** het eerste element van de betreffende collectie geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie kan de hele lijst van elementen in de geselecteerde collectie worden afgewerkt.
Indien daarna weer een **COLLECTION_NEXT**-commando wordt gegeven, heeft dit geen resultaat meer.

Om de markering weer terug te zetten naar het begin, dient het keyword **COLLECTION_RESET** gebruikt te worden.
Merk op, dat hierdoor **geen selectie** plaats vindt van het eerste element. Om het eerste element te selecteren dient na het resetten altijd **COLLECTION_NEXT** gebruikt te worden !!!

Een voorbeeld waarin het gebruik van **COLLECTION_RESET** nodig is betreft de onderstaande collectie A:

```
aaa
bbb
ccc
ddd
```

Indien het element 'ccc' in de collectie aanwezig is, dienen alle elementen uit de collectie geprint te worden.

Het volgende script doet dit **niet**:

```
if collection_find A ccc
    BEGIN
    while collection_next A
        collection_print A
    END
```

Het resultaat van dit script is:

```
ddd
```

Het element 'ccc' wordt met **COLLECTION_FIND** namelijk zowel gevonden als geselecteerd, waardoor de opdracht 'collection_next A' de markering bij 'ddd' zal plaatsen.
Om het gewenste resultaat te verkrijgen, moet het script er daarom als volgt uitzien:

```
if collection_find A ccc
    BEGIN
    collection_reset A
    while collection_next A
        collection_print A
    END
```

Dankzij het reset-commando begint de **WHILE**-lus nu wel bij het eerste element in de collectie.

Zie ook: **COLLECTION_FIND, COLLECTION_NEW, COLLECTION_NEXT, COLLECTION_ADD, COLLECTION_PRINT, COLLECTION_CLOSE, COLLECTION_DELETE.**

COLUMN

Syntax: COLUMN <positie | $positie>

Functie: positioneert de cursor op de aangegeven kolom-positie. (Lay Out)

Werking: Om alle uitvoer netjes in kolommen op papier te krijgen, is het keyword **COLUMN** de meest aangewezen weg. Uitgaande van positie 1 als eerste positie op de regel, verschuift de **COLUMN**-opdracht de cursor naar de aangegeven kolom-positie.
In combinatie met **HEADER**, **NEWLINE** OFF en **NEWLINE** ON maakt dit het vormgeven van koppen op de pagina's van een rapport tot een eenvoudige handeling.
Voorbeeld:

```
(* eerste regel *)
newline off
print Naam
column 10
print :
column 15
newline on
print SDW

(* tweede regel *)
newline off
print Module
column 10
print :
column 15
newline on
print SDWrite
```

levert als uitvoer:

```
Naam    :   SDW
Module  :   SDWrite

|       |   |
1       10  15
```

Dit werkt echter niet als er eerst meer tekens geprint worden dan het aantal dat als kolom-positie aan **COLUMN** wordt meegegeven.
Het volgende script(gedeelte):

```
newline off
print abcdefg
column 5
print xxxxx
column 15
newline on
print yyyyy
```

zal als uitvoer opleveren:

```
abcdefgxxxxx  yyyyy
```

Omdat 'xxxxx' niet op positie 5 kon beginnen (daarvoor is de eerste geprinte tekenreeks 'abcdefg' te lang) maar pas op positie 8, begint ook 'yyyyy' 3 posities verder dan de opgegeven kolom-positie, dus op positie 18 in plaats van 15.

Zie ook: **INDENT, SPACES.**

COMMENT

Syntax: COMMENT [tekst]

Functie: toevoegen van commentaar aan een script. (-)

Werking: Elke regel in een script-definitie die begint met **COMMENT** wordt beschouwd als commentaar. Op de verwerking van het script hebben dergelijke regels geen invloed.

Door commentaar aan een script toe te voegen, wordt de leesbaarheid en onderhoudbaarheid verbeterd. Om die reden wordt aangeraden elk script van het noodzakelijke commentaar te voorzien.

Zie ook: -.

DATE

Syntax: DATE

Functie: schrijft de datum naar het current uitvoerkanaal of naar een variabele. (-)

Werking: Het keyword **DATE** doet niets anders dan het naar het current uitvoerkanaal of een variabele schrijven van de datum in het formaat dd:mm:jjjj.
Het mag niet verward worden met het MS-DOS-commando DATE, waarmee ook een nieuwe datum kan worden opgegeven.

Het is mogelijk de datum als waarde aan een variabele toe te kennen.
Dit gaat als volgt:

```
set $datum DATE
```

De opdracht:

```
print $datum
```

zal nu dezelfde uitvoer opleveren als **DATE**.

Om de systeem-datum af te drukken kan tenslotte ook gebruik gemaakt worden van de **standaard SDWrite-variabelen** $SDWday, $SDWmonth en $SDWyear.
Deze variabelen worden automatisch door SDWrite gevuld. Het commando:

```
print $SDWday\-$SDWmonth\-$SDWyear
```

zal de datum schrijven in het formaat dd-mm-jjjj.

Zie ook: **PRINT**, **SET**, **TIME**.

DESCRIPTION

Syntax: DESCRIPTION <beschrijving>

Functie: opgeven van een korte beschrijving van het script. (Script)

Werking: Een selectie-window van SDW-componenten bevat standaard uitsluitend de namen van de betreffende componenten. Bij SDW-formulieren en Scripts kan hieraan voor elke naam ook een korte omschrijving van het formulier/script worden toegevoegd door in de definitie een **DESCRIPTION** op te nemen.

De tekst achter **DESCRIPTION** wordt in selectie-windows tussen vierkante haken achter de component-naam getoond.

Bijvoorbeeld: indien in de definitie van het Script 'model' het volgende commando wordt opgenomen:

```
DESCRIPTION Meta-gegevens afdrukken
```

zal dit script in selectie-windows als volgt getoond worden:

```
model [Meta-gegevens afdrukken]
```

Aangeraden wordt elke definitie van een SDW-formulier en Script van een **DESCRIPTION**-regel te voorzien.

Een **DESCRIPTION moet** als **eerste commando** in een script worden opgenomen.

Zie ook: -.

DIAGRAM_HEADER

Syntax: DIAGRAM_HEADER <ON | OFF>

Functie: aangeven dat een diagram met of zonder diagramkop afgedrukt dient te worden. (SDW)

Werking: Boven elk diagram van SDW bevindt zich een zogenaamde diagramkop. De indeling en inhoud van deze diagramkop kan per diagram-type worden bepaald met behulp van het hulpprogramma 'Kopdefinitie'.

Standaard worden diagrammen bij het afdrukken vanuit SDWrite afgedrukt **met** diagramkop. Indien dit niet gewenst is, dient voorafgaand aan het afdrukken het volgende commando gegeven te worden:

```
diagram_header off
```

Elk diagram dat na deze opdracht wordt afgedrukt, wordt afgedrukt zonder diagramkop.
Indien elders in het script het volgende commando wordt gegeven:

```
diagram_header on
```

zullen vanaf die plaats alle volgende diagrammen worden afgedrukt mèt bijbehorende diagramkop.

Zie ook: **OUTPUT, PRINT_DIAGRAM.**

↖ Definiëren van diagramkoppen
SDWorkstation 1: Kopdefinitie

DISPLAY

Syntax: DISPLAY { $varnaam | tekst }

Functie: stuurt de gegeven tekst en/of variabele(n) naar de standaard fouten-output (d.i. het beeldscherm), ongeacht het met **OUTPUT** opgegeven current uitvoerkanaal. (IO)

Werking: Het keyword **DISPLAY** is vooral bedoeld voor het opzetten van min of meer interactieve scripts.
DISPLAY kan bijvoorbeeld gebruikt worden om een vragende zin naar het scherm te sturen, waarna met behulp van het keyword **PROMPT** het antwoord kan worden ingelezen:

```
newline off
display Hoeveel deelnemers?
set $aantal prompt

print Het aantal deelnemers is $aantal
```

Met als resultaat:

```
Hoeveel deelnemers? _
```

Indien nu '12 <Enter>' wordt ingetoetst, verschijnt het volgende op het scherm:

```
Het aantal deelnemers is 12
```

Ook kan **DISPLAY** gebruikt worden om echo's van het verwerkingsproces van een script te geven. Indien een script alle uitvoer bijvoorbeeld naar een file stuurt, verloopt dat hele proces onzichtbaar voor de gebruiker. Door op belangrijk plaatsen in een script **DISPLAY**-opdrachten te plaatsen, kan het verloop van een script via het beeldscherm gevolgd worden.
Bijvoorbeeld:

```
output data.txt
display De uitvoerfile 'data.txt' wordt geopend
<etc.>
```

DISPLAY gebruikt als uitvoerkanaal altijd de standaard fouten-output, dat wil zeggen: het beeldscherm.
Indien met **OUTPUT** een ander uitvoerkanaal is geopend, wordt alle uitvoer naar dat andere uitvoerkanaal gestuurd **behalve** de uitvoer van de met **DISPLAY** gegeven schrijf-opdrachten.
Alle na een **DISPLAY**-opdracht gegeven schrijfopdrachten gaan vervolgens weer gewoon naar het met **OUTPUT** opgegeven uitvoerkanaal.

Zie ook: **OUTPUT, PROMPT.**

DIVIDE

Syntax: DIVIDE <$varnaam> <waarde | $waarde>

Functie: deelt de gegeven variabele door de gegeven waarde. (VAR)

Werking: Met **DIVIDE** wordt de rekenkundige bewerking delen uitgevoerd. Daarbij wordt uitgegaan van integers, d.w.z. gehele getallen (...,-2, -1, 0, 1, 2, ...).
De variabele dient vooraf met het **SET**-commando te zijn geïnitialiseerd. Is dat niet gebeurd, dan volgt een foutmelding en wordt de verwerking van het script afgebroken.
Indien de variabele een waarde heeft of krijgt die geen integer is (bijvoorbeeld een woord of een getal met decimalen), dan wordt het integer-gedeelte van de betreffende variabele gebruikt. Dat wil zeggen dat het gedeelte achter de komma wordt genegeerd. Er vindt derhalve **geen** afronding plaats.
Hetzelfde gebeurt met de waarde die als tweede parameter wordt meegegeven. De opdrachten:

```
divide $x 3        (x = computer)
divide $x 3        (x = 2)
divide $x 3        (x = 6,9)
divide $x 3        (x = 8,7)
divide $x 2,5      (x = 12,5)
```

hebben derhalve als resultaat:

```
x = 0              (0 : 3)
x = 0              (2 : 3 = 0,66666 = 0)
x = 2              (6 : 3)
x = 2              (8 : 3 = 2,66666 = 2)
x = 6              (12 : 2)
```

Het is ook mogelijk als tweede waarde een variabele mee te geven. Het eerste argument **moet** een variabele zijn. Is dat niet zo, dan wordt de verwerking van het script afgebroken.

Zie ook: **ADD, MULTIPLY, SET, SUBTRACT.**

DOLLAR

Syntax: DOLLAR <ON | OFF>

Functie: aan- of uitzetten van de $-substitutie. (VAR)

Werking: Tijdens het schrijven van teksten met het dollar-teken ('$') wordt door SDWrite standaard gezocht naar een variabele. Indien de betreffende variabele geïnstantieerd is, zal vervolgens de **waarde** van de variabele afgedrukt worden.
De volgende combinatie:

```
set $tekst print SDWrite
set $soort print programma
print $tekst is een $soort
```

zal derhalve standaard de volgende uitvoer opleveren:

```
SDWrite is een programma
```

Dit wordt dollar-substitutie genoemd.
Indien het gewenst is dat de variabele wordt afgedrukt en niet de waarde, dient de dollar-substitutie te worden uitgezet. Geef hiervoor het volgende commando:

```
dollar off
```

Het voorbeeld-tekstje zal nu de volgende uitvoer opleveren:

```
$tekst is een $soort
```

Deze functionaliteit kan bijvoorbeeld gebruikt worden om vanuit een script andere scripts af te drukken.

Zie ook: **PRINT**, **SET**.

EJECT

Syntax: EJECT

Functie: forceert een formfeed indien **FORMFEED** ON is gespecificeerd. (Lay Out)

Werking: De opdracht **EJECT** zorgt ervoor dat een formfeed wordt gegeven. Dit gebeurt echter alleen wanneer **FORMFEED** ON staat.
Staat **FORMFEED** OFF dan zal het commando **EJECT** geen gevolg hebben.
Een voor de hand liggende plaats voor het **EJECT**-commando is de eerste regel van de header. Door op die plaats een formfeed te geven, wordt ervoor gezorgd dat de header steeds op een nieuwe pagina wordt geprint.
Om te voorkomen dat ook de eerste header reeds een formfeed oplevert (wat neerkomt op het ongebruikt laten van een pagina), dient het commando **FORMFEED** ON in de header te worden opgenomen **na** het **EJECT**-commando.

Bijvoorbeeld:

```
HEADER
    BEGIN
    eject
    print System Development Workbench
    print Systeem : test
    print - - - - - - - - - - - - - - -
    formfeed on
    END
```

De eerste keer dat deze header wordt aangeroepen staat **FORMFEED** nog OFF (default-instelling). Het commando **EJECT** heeft dan dus geen formfeed tot gevolg.
Aan het einde van de eerste header-uitvoering wordt **FORMFEED** ON gezet, zodat in de volgende header-aanroep(en) met **EJECT** wèl een formfeed wordt gerealiseerd (ervan uitgaande dat **FORMFEED** niet elders in het script weer OFF wordt gezet).

Zie ook: **FORMFEED, PAGELENGTH.**

ELSE

Syntax: IF <conditie> <actie1> ELSE <actie2>

Functie: geeft in een **IF**-constructie aan wat er moet gebeuren als niet aan de voorwaarde wordt voldaan. (PRG)

Werking: Het keyword **ELSE** kan uitsluitend gebruikt worden in combinatie met **IF**, en dient om de alternatieve actie aan te geven, d.w.z. de actie die moet worden uitgevoerd indien niet is voldaan aan de voorwaarde achter **IF**.

De actie achter **ELSE** kan zelf weer bestaan uit een **IF**- **ELSE**-constructie, enzovoorts, waardoor zeer uitgebreide ALS-DAN-ANDERS-constructies mogelijk zijn.
Bijvoorbeeld:

```
display Uitvoer naar file of naar printer (f/p) ?
set $antwoord prompt
if equal $antwoord f
    BEGIN
    display Filenaam :
    set $out prompt
    END
else
if equal $antwoord p
    BEGIN
    display Printerpoort LPT1 of LPT2 (1/2)?
    set $poort prompt
    if equal $poort 1
        set $out lpt1
    else
        set $out lpt2
    END
else
    display Onbekende keuze!
```

In dit script-gedeelte wordt gevraagd of de uitvoer naar file of naar een printer moet. Wordt gekozen voor file ('f'), dan wordt gevraagd om de filenaam. Wordt gekozen voor printer ('p'), dan wordt gevraagd naar de gewenste printerpoort. Wordt noch een 'f' noch een 'p' ingetoetst, dan verschijnt een foutmelding.

Zie ook: **BEGIN, END, IF, WHILE.**

END

Syntax: END

Functie: duidt het einde aan van een serie bij elkaar horende opdrachten. (PRG)

Werking: Het keyword **END** duidt op zichzelf geen handeling aan. Het dient altijd gebruikt te worden in combinatie met **BEGIN** en een aantal andere statements die tussen **BEGIN** en **END** in moeten staan.
De aanduiding **BEGIN** en **END** om een aantal statements betekent, dat die statements bij elkaar horen.
Dit is vooral van belang in constructies met **IF** en **WHILE**, maar ook bij het maken van een **HEADER** en een **MACRO**.
Vergelijk de volgende twee script-gedeeltes:

(A.)
```
if equal $var 12
    print regel1
    print regel2
```

(B.)
```
if equal $var 12
    BEGIN
    print regel1
    print regel2
    END
```

Ervan uitgaande dat $var inderdaad gelijk is aan 12 zullen beide scripts dezelfde uitvoer opleveren. Script A print regel1 omdat aan de voorwaarde in het **IF**-statement wordt voldaan, en regel2 omdat het tweede print-statement niet met de **IF**-constructie verbonden is en derhalve altijd geprint wordt.
Script B print zowel regel1 als regel2 omdat aan de voorwaarde in het **IF**-statement wordt voldaan.

Indien $var **niet** gelijk is aan 12, leveren de scripts echter niet dezelfde uitvoer.

Script A zal regel1 niet printen, omdat niet aan de voorwaarde

wordt voldaan. Maar regel2 wordt door script A wèl geprint, omdat dat een gewoon print-statement is.

In script B zijn echter beide print-statements verbonden aan de **IF**-constructie, omdat ze tussen **BEGIN** en **END** staan. Script B levert in dit geval derhalve géén uitvoer.

Overigens zou het verschil tussen beide scripts door een iets andere lay out van script A al veel duidelijker zijn uitgekomen. Vergelijk:

```
(A'.)
if equal $var 12
    print regel1
print regel2
```

Door alleen de bij de **IF**-constructie behorende print-opdracht te laten inspringen, wordt duidelijk dat de tweede print-opdracht van een andere aard is.
Dit is overigens niets meer dan een goede lay out-gewoonte: voor de verwerking van een script doet het al dan niet inspringen niet terzake. De SDWrite-interpreter zal inspringende regels niet anders vertalen dan niet inspringende !

In combinatie met **MACRO** en **HEADER** zijn **BEGIN** en **END** beslissend voor welke opdrachten als onderdeel van de macro respectievelijk header worden gezien.
De opbouw van een macro-definitie en een header-definitie zijn hetzelfde:

```
HEADER
    <definitie>
```

en:

```
MACRO <naam>
    <definitie>
```

waarbij de definitie kan bestaan uit:

```
BEGIN
<opdracht(en)>
END
```

of:

<opdracht> (maximaal 1)

Indien de definitie van de macro of header uit meerdere opdrachten bestaat, dan **moet** gebruik gemaakt worden van **BEGIN** en **END**.
Wordt dat niet gedaan, dan zal alleen de eerste opdracht onder 'HEADER' of 'MACRO <naam>' als definitie beschouwd worden.
BEGIN en **END** werken hier dus hetzelfde als in het voorbeeld hiervoor.

Zie ook: **BEGIN, IF, HEADER, MACRO, WHILE.**

EQUAL

Syntax: EQUAL <waarde1|$var1> <waarde2|$var2>

Functie: vergelijkt de gegeven variabele met de gegeven waarde of variabele. (VAR)

Werking: Met het **EQUAL**-commando worden twee waarden met elkaar vergeleken, met als enige mogelijke uitkomsten 'true' of 'false', d.w.z. dat de te vergelijken waarden gelijk of ongelijk zijn.
Het keyword **EQUAL** is alleen zinvol in combinatie met **IF** of **WHILE**.

De opdracht:

```
equal $var 12
```

zal geen ander resultaat dan een foutmelding geven.
Zinvol wordt het **EQUAL**-commando pas als wordt aangegeven wat er moet gebeuren indien de vergeleken waarden gelijk of ongelijk zijn. Vereist is derhalve een van de constructies:

als <voorwaarde> dan <actie>

of

zolang <voorwaarde> dan <actie>

Dit leidt tot vier mogelijke SDWrite-constructies met **EQUAL**:

- if equal <waarde1> <waarde2>
  <actie>
- if NOT equal <waarde1> <waarde2>
  <actie>
- while equal <waarde1> <waarde2>
  <actie>
- while NOT equal <waarde1> <waarde2>
  <actie>

Zie ook: **GREATER, GREATEREQUAL, IF, NOT, WHILE.**

FILE

Syntax: FILE <filenaam | $filenaam>

Functie: leest en print de opgegeven file geheel, waarbij zaken als pagina-nummering en headers gehandhaafd blijven. (IO)

Werking: Voor het opnemen van een bepaalde ASCII-file in een rapport of overzicht, kan het commando **FILE** gebruikt worden, onder opgave van de ASCII-filenaam (eventueel in een variabele).
Het **FILE**-commando zorgt ervoor dat de complete file wordt weggeschreven naar het geselecteerde uitvoerkanaal.
Pagina-nummering en header worden daarbij gehandhaafd.

Indien bijvoorbeeld wordt uitgegaan van de file 'test.txt':

```
Dit is een voorbeeldtekst ten behoeve van
SDWrite.
Aan de hand van deze tekst wordt getoond
dat het FILE-commando geen invloed heeft
op de werking van HEADER en PAGENO.
```

en het script:

```
pagelength 7

HEADER
BEGIN
    print - - - - - - -
    print Test-versie
    pageno
    print - - - - - - -
END

print_header
file test.txt
```

ontstaat de uitvoer:

```
- - - - - - -
Test-versie
1
- - - - - - -
```

```
Dit is een voorbeeldtekst ten behoeve van
SDWrite.
Aan de hand van deze tekst wordt getoond
- - - - - - -
Test-versie
2
- - - - - - -
dat het FILE-commando geen invloed heeft
op de werking van HEADER en PAGENO.
```

Na de derde regel van de met **FILE** opgeroepen ASCII-file wordt de header geprint omdat op dat moment 7 regels zijn weggeschreven (vier van de eerste header plus drie van 'test.txt'). Daarmee is de opgegeven pagina-lengte (=7) bereikt, waarna automatisch de tweede header wordt geprint.
In deze tweede header is het pagina-nummer 2.

Zie ook: **HEADER, PAGENO, PRINT_HEADER.**

FORMFEED

Syntax: FORMFEED <ON | OFF>

Functie: ON zorgt ervoor dat het commando **EJECT** vóór het printen van de header een formfeed genereert. (Lay Out)

Werking: Het commando **FORMFEED** (ON of OFF) heeft geen directe gevolgen voor de uitvoer van een script. Het heeft uitsluitend gevolgen in combinatie met het commando **EJECT**.
Indien SDWrite het commando **EJECT** tegenkomt (het sein om een fysieke formfeed te geven), wordt gekeken of **FORMFEED** ON of OFF staat. Staat **FORMFEED** ON dan wordt daadwerkelijk een formfeed gegeven; staat **FORMFEED** daarentegen OFF dan wordt het **EJECT**-commando genegeerd.
De meest voorkomende plaats voor het **FORMFEED**-commando (tezamen met **EJECT**) is de header.
Bijvoorbeeld:

```
HEADER
BEGIN
    eject
    print SDW - Rapportage
    newline off
    print Pagina :
    newline on
    pageno
    print - - - - - - - - -
    eject
    formfeed on
END
```

Door het **EJECT**-commando te geven vóór **FORMFEED** ON wordt ervoor gezorgd dat niet eerst een lege pagina wordt geprint. De eerste keer dat SDWrite het **EJECT**-commando tegenkomt in dit voorbeeldscript, staat **FORMFEED** namelijk nog OFF (default), zodat die eerste keer geen formfeed wordt gegeven. Alle volgende keren zal **EJECT** wèl tot een daadwerkelijke formfeed leiden (ervan uitgaande dat **FORMFEED** niet tussentijds weer OFF wordt gezet).

Er dient rekening mee gehouden te worden, dat ook de **printer**

zodanig ingesteld kan zijn dat na een aantal regels een formfeed wordt gegeven. Op deze instelling heeft **FORMFEED** OFF geen invloed. Zie hiervoor de handleiding van de printer.

Zie ook: **EJECT, HEADER, PAGELENGTH.**

GREATER

Syntax: GREATER <waarde1|$var1> <waarde2|$var2>

Functie: bekijkt of het eerste argument groter is dan het tweede. (VAR)

Werking: Met het **GREATER**-commando worden twee waarden met elkaar vergeleken, met als enige mogelijke uitkomsten 'true' of 'false', d.w.z. dat de eerste waarde groter is dan de tweede of niet.
Het keyword **GREATER** is alleen zinvol in combinatie met **IF** of **WHILE**.

De opdracht:

```
greater $var 12
```

zal geen ander resultaat dan een foutmelding geven.
Zinvol wordt het **GREATER**-commando pas als wordt aangegeven wat er moet gebeuren indien de vergeleken waarden gelijk of ongelijk zijn. Vereist is derhalve een van de constructies:

als <voorwaarde> dan <actie>

of

zolang <voorwaarde> dan <actie>

Dit leidt tot vier mogelijke SDWrite-constructies met **GREATER**:

- if greater <waarde1> <waarde2>
  <actie>
- if NOT greater <waarde1> <waarde2>
  <actie>
- while greater <waarde1> <waarde2>
  <actie>
- while NOT greater <waarde1> <waarde2>
  <actie>

De vergelijking die met **GREATER** wordt uitgevoerd kan betrekking hebben op **getallen of teksten**. Dit is afhankelijk van de argumenten. Als beide argumenten numeriek zijn, wordt vergeleken op getalswaarde; in alle andere gevallen wordt een vergelijking uitgevoerd op basis van alfabetische volgorde.
Derhalve leiden de volgende commando's beide tot de uitvoer 'groter':

```
if greater 12 4
    print groter

if greater boot aap
    print groter
```

Zie ook: **EQUAL, GREATEREQUAL, IF, NOT, WHILE.**

GREATEREQUAL

Syntax: GREATEREQUAL <waarde1|$var1> <waarde2|$var2>

Functie: bekijkt of het eerste argument groter is dan het tweede of daaraan gelijk. (VAR)

Werking: Met het **GREATEREQUAL**-commando worden twee waarden met elkaar vergeleken, met als enige mogelijke uitkomsten 'true' of 'false', d.w.z. dat het eerste argument groter of gelijk is aan het tweede, of niet.
Het keyword **GREATEREQUAL** is alleen zinvol in combinatie met **IF** of **WHILE**. De opdracht:

```
greaterequal $var 12
```

zal geen ander resultaat dan een foutmelding geven.
Zinvol wordt het **GREATEREQUAL**-commando pas als wordt aangegeven wat er moet gebeuren indien de vergeleken waarden gelijk of ongelijk zijn. Vereist is derhalve een van de constructies:

als <voorwaarde> dan <actie>

of

zolang <voorwaarde> dan <actie>

Dit leidt tot vier mogelijke SDWrite-constructies met **GREATEREQUAL**:

- if greaterequal <waarde1> <waarde2>
  <actie>
- if NOT greaterequal <waarde1> <waarde2>
  <actie>
- while greaterequal <waarde1> <waarde2>
  <actie>
- while NOT greaterequal <waarde1> <waarde2>
  <actie>

De vergelijking die met **GREATEREQUAL** wordt uitgevoerd kan betrekking hebben op **getallen of teksten**. Dit is afhankelijk van de argumenten. Als beide argumenten numeriek zijn, wordt vergeleken op getalswaarde; in alle andere gevallen wordt een vergelijking uitgevoerd op basis van alfabetische volgorde.
Derhalve leiden de volgende commando's beide tot de uitvoer 'grotergelijk':

```
if greaterequal 12 4
    print grotergelijk

if greaterequal boot aap
    print grotergelijk
```

Zie ook: **EQUAL, GREATER, IF, NOT, WHILE.**

HAS_GENERATED_ITEM_TEXT

Syntax: HAS_GENERATED_ITEM_TEXT [naam | $naam]

Functie: geeft aan of de geselecteerde rubriek gevuld is of niet. (SDW)

Werking: Met **HAS_GENERATED_ITEM_TEXT** kan gecontroleerd worden of een bepaalde gegenereerde rubriek gevuld is. De rubriek die gecontroleerd wordt, is de als argument opgegeven rubriek òf de gegenereerde rubriek die met **SELECT_GENERATED_ITEM** of **NEXT_GENERATED_ITEM** is geselecteerd.
Het commando is uitsluitend zinvol in combinatie met **IF**.

De twee volgende constructies hebben hetzelfde resultaat:

(A.)
```
select_generated_item Markering
if NOT has_generated_item_text
    print Component niet gemarkeerd
```

(B.)
```
if NOT has_item_text Markering
    print Component niet gemarkeerd
```

Zie ook: **SELECT_GENERATED_ITEM, NEXT_GENERATED_ITEM, IF, HAS_ITEM_TEXT.**

HAS_ITEM_TEXT

Syntax: HAS_ITEM_TEXT [rubrieksnaam | $rubrieksnaam]

Functie: geeft aan of de geselecteerde rubriek gevuld is of niet. (SDW)

Werking: Met **HAS_ITEM_TEXT** kan eenvoudig gecontroleerd worden of een bepaalde rubriek gevuld is. De rubriek die gecontroleerd wordt, is de als argument opgegeven rubriek òf de met met **SELECT_ITEM** of **NEXT_ITEM** geselecteerde rubriek.
Het commando is uitsluitend zinvol in combinatie met **IF**.

De twee volgende constructies hebben hetzelfde resultaat:

(A.)
```
select_item beschrijving
if NOT has_item_text
    print Beschrijving niet ingevuld
```

(B.)
```
if NOT has_item_text beschrijving
    print Beschrijving niet ingevuld
```

Zie ook: **SELECT_ITEM, NEXT_ITEM, IF, HAS_GENERATED_ITEM_TEXT.**

HEADER

Syntax: HEADER

Functie: geeft aan dat het volgende commando (of een reeks commando's tussen **BEGIN** en **END**) moet worden uitgevoerd indien het aantal regels dat op een blad past, wordt overschreden, of indien het **PRINT_HEADER**-commando wordt gegeven. (Lay Out)

Werking: Met het keyword **HEADER** wordt het begin aangegeven van een kopdefinitie.
Een kop kan bovenaan elke pagina worden geprint, maar ook op andere plaatsen.
Een kopdefinitie kan bestaan uit één commando of uit een reeks commando's tussen **BEGIN** en **END**.
Voorbeelden:

```
(* header A *)
    HEADER
        eject

(* header B *)
    HEADER
    BEGIN
        eject
        print System Development Workbench
        print SDWrite-rapportage
        newline off
        print Pagina :
        newline on
        pageno
        formfeed on
    END
```

In een script mag maximaal één header voorkomen, maar een header is niet verplicht.

De meest aangewezen plaats voor de kopdefinitie is het begin van een script. Dit leidt er echter **niet** toe dat direct aan het begin de **HEADER**-commando's worden uitgevoerd.

De SDWrite-interpreter voert de opdracht(en) onder **HEADER** alleen uit indien:

- het aantal regels dat op de pagina past (de pagelength) wordt overschreden
- het commando **PRINT_HEADER** wordt gegeven
- aan het keyword **NEED** een getal wordt meegegeven dat groter is dan het aantal regels dat op de betreffende pagina nog resteert

Elke keer dat aan een van deze voorwaarden wordt voldaan, zullen automatisch alle commando's die direct onder **HEADER** staan, worden uitgevoerd.
Dit gebeurt op geen enkel ander moment.

Het schrijven van de header houdt niet automatisch in dat ook een formfeed wordt gegeven.
Indien dat wel de bedoeling is, dient dit in de header te worden aangegeven met de daarvoor bedoelde commando's (**FORMFEED** en **EJECT**).

Zie ook: **EJECT, FORMFEED, NEED, PAGELENGTH.**

IF

Syntax: IF <conditie> <actie>

Functie: voert de gegeven actie (een commando of een reeks commando's tussen **BEGIN** en **END**) uit indien aan de gegeven conditie wordt voldaan. (PRG)

Werking: Met een **IF**-constructie wordt het uitvoeren van een bepaalde ACTIE (een commando of een reeks commando's tussen **BEGIN** en **END**) gebonden aan een bepaalde voorwaarde.
Wordt aan de betreffende voorwaarde voldaan, dan wordt de betreffende actie uitgevoerd; wordt niet aan de voorwaarde voldaan, dan wordt de betreffende actie niet uitgevoerd.

Een voorbeeld:

```
newline off
display Doorgaan? (j/n)
set $antwoord prompt
newline on

if equal $antwoord n
    BEGIN
    print
    print ... user-interrupt ...
    print End of run
    END
```

In dit voorbeeld wordt de gebruiker gevraagd of moeten worden doorgegaan of niet. Toetst de gebruiker in dat niet moet worden doorgegaan ('n'), dan zal de tekst achter **IF** naar het current uitvoerkanaal geschreven worden.
Er vanuit gaande dat dit het scherm is, zal derhalve het volgende op het scherm zichtbaar zijn:

```
Doorgaan? (j/n) n

... user-interrupt ...
End of run
```

Wat er gebeurt als de gebruiker een ander antwoord dan 'n'

geeft, hangt af van de rest van het script.

Het is mogelijk een **IF**-constructie uit te breiden met een **ELSE**-gedeelte. Daardoor ontstaat een ALS-DAN-ANDERS-constructie.

Zie ook: **BEGIN, ELSE, END, WHILE.**

INDENT

Syntax: INDENT <positie | $positie>

Functie: inspringen tot de aangegeven kolom-positie. (Lay Out)

Werking: Om een aantal direct op elkaar volgende regels steeds op dezelfde kolom-positie te laten beginnen, kan gebruik gemaakt worden van het keyword **INDENT**.
Om bijvoorbeeld de tweede, derde en vierde regel van de uitvoer 4 posities te laten inspringen ten opzichte van de rest van de tekst, kunnen de volgende commando's worden gebruikt:

```
print regel1
indent 4
print regel2
print regel3
print regel4
indent 0
print regel5
```

hetgeen de volgende uitvoer oplevert:

```
regel1
    regel2
    regel3
    regel4
regel5
```

Zoals uit dit voorbeeld reeds blijkt, kan **INDENT** weer worden 'uitgezet' door het commando 'indent 0'.

Zie ook: **COLUMN, SPACES.**

LENGTH

Syntax: LENGTH <string | $string>

Functie: geeft de string-lengte van het argument. (Lay Out)

Werking: Met het commando **LENGTH** kan de lengte van elke willekeurige tekst of variabele worden achterhaald.
Bijvoorbeeld:

```
length user
```

heeft als resultaat: 4.

Zoals elk schrijf-commando kan ook het **LENGTH**-commando worden gebruikt in combinatie met **SET** om zodoende de lengte van een string als waarde aan een variabele mee te geven.

Uitgaande van de volgende opdracht:

```
set $tekst SDWrite
```

levert:

```
set $tekstlengte length $tekst
print $tekstlengte
```

derhalve als uitvoer: 7.

Zie ook: **LINE, PRINT_PART, SET, SUBSTRING.**

LINE

Syntax: LINE [waarde | $waarde]

Functie: leest een regel (in zijn geheel of vanaf de opgegeven waarde) uit het geopende bestand en stuurt deze naar het uitvoer-device of een variabele. (IO)

Werking: Met het commando **LINE** wordt een regel (of een gedeelte daarvan) uit een file ingelezen en gestuurd naar het current uitvoerkanaal of een variabele. De invoerfile dient hiervoor eerst met **LINE_OPEN** geopend te zijn.
Gesteld dat voor de rapportage over een SDW-systeem steeds wisselende component-typen van belang zijn (niet alle component-typen, en ook niet steeds dezelfde), dan kan met behulp van het **LINE**-commando met slechts één script aan de vraag voldaan worden.
Daarvoor dienen de betreffende component-typen voor één bepaald rapport opgenomen te worden in een file, bijvoorbeeld 'typen.txt', die er als volgt uit kan zien:

```
Functie
Stroom
Buffer
Gegevenselement
```

en die vervolgens met **LINE_OPEN** en **LINE** ingelezen wordt in een script met (o.a.) de volgende commando's:

```
line_open typen.txt                    (* a *)

while set $type line                   (* b *)
    BEGIN
    select_component_type $type        (* c *)
    <* gewenste acties met het component-type *>
    END
```

Nadat de invoerfile geopend is (a), wordt de eerste ingelezen regel als waarde toegekend aan de variabele $type (b), waarna het betreffende component-type geselecteerd wordt (c). De eerste keer wordt dus het component-type Functie geselecteerd.

Daarna worden op dezelfde wijze ook de volgende component-typen ingelezen en verwerkt, totdat het einde van de file bereikt is en het proces (bij b) wordt gestopt.
Door de volgende keer aan 'typen.txt' een andere inhoud te geven, zal hetzelfde script nu andere component-typen verwerken.
Het spreekt voor zichzelf dat hetzelfde ook mogelijk is voor het selecteren van specifieke componenten en rubrieken.

Zelfs zeer specifieke combinaties zijn op deze wijze te benaderen. Stel dat slechts twee rubrieken van twee componenten van belang zijn voor een speciaal rapport, bijvoorbeeld de rubriek 'beschrijving' van de Stroom 'invoer' en de rubriek 'functionaris' van de Functie 'invoeren', dan kan deze specifieke informatie aan een script worden meegegeven met behulp van een ASCII-file met de volgende inhoud:

```
Stroom
invoer
beschrijving
Functie
invoeren
functionaris
```

Het bijbehorende script leest uit deze ASCII-file (bijv. 'names.txt' genaamd) de betreffende namen in, en wel als volgt:

```
line_open names.txt

while set $type line                    (* 1 *)
    BEGIN
    select_component_type $type         (* 1a *
    set $comp line                      (* 2 *)
    select_component $comp              (* 2a *)
    set $item line                      (* 3 *)
    select_item $item                   (* 3a *)
    <* gewenste handelingen met de rubriek *>
    END
```

Op deze wijze worden achtereenvolgens het component-type (1), de componentnaam (2) en de rubrieksnaam (3) ingelezen en geselecteerd (1a, 2a, 3a), waarna datzelfde proces zich herhaalt voor alle volgende in de ASCII-file opgenomen

namen.

Door een parameter mee te geven aan het **LINE**-commando wordt ervoor gezorgd dat niet de hele ingelezen regel wordt weggeschreven naar het current uitvoerkanaal of een variabele, maar slechts het deel vanaf de opgegeven positie.

Uitgaande van een invoerfile 'invoer.txt' met als inhoud:

```
001;Functie
002;Stroom
003;Buffer
004;Gegevenselement
```

zou derhalve regel b uit het eerste voorbeeld in deze beschrijving, vervangen moeten worden door de regel:

```
while set $type line 5              (* b *)
```

De eerste vier tekens van de invoerfile worden hierdoor wel ingelezen, maar niet weggeschreven naar de variabele $type. Aan $type worden zodoende ook in dit geval achtereenvolgens de waarden 'Functie', 'Stroom', 'Buffer' en 'Gegevens-element' toegekend.

Zie ook: **LINE_OPEN, PRINT_PART.**

LINE_CLOSE

Syntax: LINE_CLOSE

Functie: afsluiten van de huidige invoerfile. (IO)

Werking: Een met **LINE_OPEN** geopende file kan gesloten worden met het commando **LINE_CLOSE**.
Dit keyword heeft geen argument, aangezien er altijd maar één file tegelijk geopend kan worden.

Aangeraden wordt om in alle gevallen waarin een file wordt gelezen, het commando **LINE_CLOSE** te gebruiken zodra het lezen is afgerond.

Zie ook: **LINE_OPEN**, **LINE**.

LINE_OPEN

Syntax: LINE_OPEN <filenaam | $filenaam>

Functie: opent het gegeven bestand voor verwerking. (IO)

Werking: Elke denkbare ASCII-file kan met het commando **LINE_OPEN** geopend worden voor verwerking in een SDWrite-script. Het eigenlijke inlezen wordt vervolgens gedaan met het commando **LINE**.

Bij het openen van de invoerfile dient rekening gehouden te worden met drive-aanduiding en pad voor de betreffende file.

Tevens dient erop gelet te worden dat een file niet tegelijkertijd als invoerfile (met **LINE_OPEN**) èn als uitvoerfile (met **OUTPUT**) is geopend.
De combinatie:

```
output tekst.txt
line_open tekst.txt
```

levert weliswaar geen foutmelding op, maar kan wel leiden tot ongewenste resultaten.

Het is ook mogelijk een **variabele** als argument mee te geven. Deze variabele dient vooraf met **SET** een waarde gekregen te hebben.

Zie ook: **FILE, LINE, OUTPUT.**

MACRO

Syntax: MACRO <naam | $naam>

Functie:
- eerste voorkomen: onthoudt onder de opgegeven naam het volgende commando (of een reeks commando's tussen **BEGIN** en **END**)
- alle volgende voorkomens: voert onder de opgegeven naam onthouden commando('s) uit.

(PRG)

Werking: Indien een bepaalde reeks commando's (of één bepaald commando) meerdere keren in een script dient te worden uitgevoerd, kan deze reeks commando's worden opgenomen in een zogenaamde macro.
Om de betreffende commando's op een bepaalde plaats uit te voeren, hoeft in het script dan niet de hele reeks commando's opnieuw te worden opgegeven, maar kan volstaan worden met het aanroepen van de macro.
De macro dient daarvoor wel eerst gedefinieerd te zijn.

Een macro-definitie kan er als volgt uitzien:

```
MACRO comp
BEGIN
    newline off
    print Componentnaam
    column 30
    print :
    column 32
    newline on
    print $comp
END
```

Met deze macro wordt een componentnaam (die eerder is toegekend aan de variabele $comp) in een vast formaat weggeschreven naar het current uitvoerkanaal.
Om deze macro uit te voeren, dient op de gewenste plaats in het script het volgende commando gegeven te worden:

```
macro comp
```

Op het moment dat de SDWrite-interpreter deze regel bereikt, zullen de commando's in de bijbehorende macro-definitie worden uitgevoerd, alsof op deze plaats in het script die commando's staan in plaats van het commando 'macro comp'.

De definitie van een macro **moet** in een script **voor** de eerste aanroep staan. SDWrite beschouwt de eerste keer dat een bepaald macro-commando wordt gegeven namelijk altijd als macro-definitie, en alle volgende keren als macro-aanroep.
Indien de macro-definitie abusievelijk **na** de eerste aanroep in een script wordt opgenomen, dan zal:

- het commando (of de reeks commando's tussen **BEGIN** en **END**) direct achter de te vroeg geplaatste macro-aanroep beschouwd worden als macro-definitie

- de beginregel van de bedoelde macro-definitie beschouwd worden als macro-aanroep, en wordt het commando (of de reeks commando's) direct achter de te vroeg geplaatste eerste macro-aanroep uitgevoerd

Het is mogelijk meerdere macro's in een script te gebruiken.
Tevens is het mogelijk een macro in een andere macro aan te roepen. Ook hierbij dient er op gelet te worden dat de geneste macro wordt gedefinieerd vóór de macro waarin zij wordt aangeroepen (vergelijkbaar met hetgeen hiervoor is beschreven).

Door gebruik te maken van een variabele, kan het starten van macro's worden gestuurd.
Bijvoorbeeld: in een script komen vier macro's voor, die afhankelijk van de door de gebruiker ingevoerde uitvoerbestemming een aantal initialisaties uitvoeren. In plaats van:

```
if equal $uitvoer printer1
    macro printer1
else
if equal $uitvoer printer2
    macro printer2
else
    macro scherm
```

kan derhalve ook het volgende commando gebruikt worden:

```
macro $uitvoer
```

Het gebruik van variabelen dient te worden beperkt tot de **MACRO**-regels waarmee macro's worden uitgevoerd. Gebruik **geen variabele** als argument **bij de definitie** van een macro.

Zie ook: **BEGIN, END, HEADER.**

MULTIPLY

Syntax: MULTIPLY <$varnaam> <waarde | $waarde>

Functie: vermenigvuldigt de gegeven variabele met de gegeven waarde. (VAR)

Werking: Met **MULTIPLY** wordt de rekenkundige bewerking vermenigvuldigen uitgevoerd. Daarbij wordt uitgegaan van integers, d.w.z. gehele getallen (...,-2, -1, 0, 1, 2, ...).
De variabele dient vooraf met het **SET**-commando te zijn geïnitialiseerd. Is dat niet gebeurd, dan volgt een foutmelding en wordt de verwerking van het script afgebroken.
Indien de variabele een waarde heeft of krijgt die geen integer is (bijvoorbeeld een woord of een getal met decimalen), dan wordt het integer-gedeelte van de betreffende variabele gebruikt. Dat wil zeggen dat het gedeelte achter de komma wordt genegeerd. Er vindt derhalve **geen** afronding plaats.
Hetzelfde gebeurt met de waarde die als tweede parameter wordt meegegeven. De opdrachten:

```
multiply $x 3       (x = computer)
multiply $x 3       (x = 2,1)
multiply $x 3,6     (x = 2,9)
multiply $x 3,6     (x = 2)
multiply $x 0,2     (x = 2)
```

hebben derhalve als resultaat:

```
x = 0               (0 * 3)
x = 6               (2 * 3)
x = 6               (2 * 3)
x = 6               (2 * 3)
x = 0               (2 * 0)
```

Het is ook mogelijk als tweede waarde een variabele mee te geven. Het eerste argument **moet** een variabele zijn. Is dat niet zo, dan wordt de verwerking van het script afgebroken.

Zie ook: **ADD, DIVIDE, SET, SUBTRACT.**

NEED

Syntax: NEED <aantal | $aantal>

Functie: checkt of het gegeven aantal regels nog op de pagina past (uitgaande van de met **PAGELENGTH** opgegeven pagina-lengte), en print de header indien dat niet zo is. (Lay Out)

Werking: Default zal SDWrite alle uitvoer van een script achter elkaar wegschrijven naar het current uitvoerkanaal. De overgang naar een andere pagina wordt daarbij door niets anders bepaald dan door de gegenereerde formfeeds. Deze kunnen worden gegenereerd door SDWrite (**EJECT** en **FORMFEED** ON) of door de printer (veelal genereren printers na een bepaald aantal regels automatisch een formfeed, zelfs als **FORMFEED** in SDWrite OFF staat; zie hiervoor de printer-handleiding).
Het is echter niet in alle gevallen van te voren aan te geven, waar een formfeed gewenst zal zijn. Soms is dit namelijk afhankelijk van het aantal regels dat nog geschreven moet worden ten opzichte van het aantal regels dat nog op de pagina past.
Het kan bijvoorbeeld gewenst zijn dat de inhoud van een rubriek in zijn geheel op één pagina wordt afgedrukt en niet wordt verdeeld over twee pagina's. In dat geval is alleen een formfeed noodzakelijk indien er op de pagina onvoldoende regels resteren om de volledige tekst weg te schrijven.

Met **NEED** biedt SDWrite de mogelijkheid aan dit soort wensen tegemoet te komen.
Het **NEED**-commando ziet er als volgt uit:

```
need 12
```

en werkt op de volgende wijze:

- op het moment dat het **NEED**-commando wordt gegeven, wordt het aantal reeds op deze pagina geschreven regels vergeleken met de pagina-lengte. Daarmee wordt niet de werkelijke lengte van een pagina bedoeld, maar de lengte die is opgegeven met het SDWrite-commando

PAGELENGTH !

- indien het verschil tussen deze pagina-lengte en het aantal reeds geschreven regels **groter of gelijk** is aan het getal dat aan het **NEED**-commando is meegegeven, dan heeft het **NEED**-commando geen gevolg en wordt het script gewoon vervolgd.
  Is het verschil echter **kleiner** dan het getal dat aan het **NEED**-commando is meegegeven, dan wordt eerst de **HEADER** gegenereerd alvorens het script wordt vervolgd. In deze **HEADER** kan de gewenste formfeed zijn opgenomen, maar dit is niet noodzakelijk.

Een voorbeeld:
In een rapport dient de rubriek 'definitie' van een bepaalde component in zijn geheel op één pagina weggeschreven te worden. Deze rubriek bevat altijd slechts 1 regel tekst, zodat tezamen met de rubrieksnaam en de rubrieksheader 3 regels nodig zijn.
In dat geval zal het volgende script-gedeelte:

```
need 3
print_item_name
print_item_header
print_item_text
```

een van de volgende twee mogelijke resultaten hebben:

- indien er (uitgaande van de opgegeven pagina-lengte) nog 3 of meer regels op de pagina passen, zullen meteen de drie print-opdrachten worden uitgevoerd;

- indien er (uitgaande van de opgegeven pagina-lengte) minder dan 3 regels over zijn, wordt de **HEADER** geprint en worden de drie print-opdrachten pas daarna uitgevoerd.

Het is hierbij niet noodzakelijk dat de rubriek daadwerkelijk op een nieuwe fysieke pagina wordt weggeschreven! Dat gebeurt alleen indien in de header (of door de printer) een formfeed wordt gegenereerd!

Zie ook: **HEADER, PAGELENGTH.**

NEWLINE

Syntax: NEWLINE [ON | OFF]

Functie: geeft na elke print-opdracht een newline (ON), print alle print-opdrachten op dezelfde regel (OFF), of voert een newline uit (geen parameter). (Lay Out)

Werking: Default genereert SDWrite na elke schrijf-opdracht een newline. Dit geldt niet alleen voor uitvoer die met een van de **PRINT**-commando's wordt weggeschreven (zoals **PRINT** en **PRINT_COMPONENT**) maar ook voor andere uitvoer-commando's (zoals **DATE**, **DISPLAY** en **TIME**).
Alleen regels die eindigen op een puntkomma worden default geschreven zonder newline. Zie hiervoor het keyword **SEMICOLON**.

Het kan echter gewenst zijn dat meerdere schrijf-opdrachten op dezelfde regel terechtkomen. In dat geval dient **NEWLINE** OFF gezet te worden totdat weer een newline gewenst is. Op dat moment kan **NEWLINE** weer ON worden gezet. Dit dient te gebeuren vóór de laatste schrijf-opdracht die nog op dezelfde regel moet worden geschreven.
Bijvoorbeeld:

```
newline off
print Datum
column 10
print :
spaces 2
---->              (*1*)
date
---->              (*2*)
print SDWrite-rapport
print nummer 1
```

Indien de opdracht **NEWLINE** ON wordt gegeven op positie (*1*), dan ziet de uitvoer er als volgt uit:

```
Datum      :   10-04-1993
SDWrite-rapport
nummer 1
```

Met **NEWLINE** ON daarentegen op positie (*2*), wordt pas na 'SDWrite-rapport' de eerste newline gegenereerd:

```
Datum      :  10-04-1993SDWrite-rapport
nummer 1
```

Het commando **NEWLINE** zónder parameter genereert altijd een newline, ongeacht of **NEWLINE** OFF dan wel ON staat.

Zie ook: **SEMICOLON**.

NEXT_COMPONENT

Syntax: NEXT_COMPONENT

Functie: selecteert de volgende component van het geselecteerde component-type uit de SE. (SDW)

Werking: Om gegevens over een component te kunnen benaderen, dient deze component eerst geselecteerd te worden. Dit kan met het commando **SELECT_COMPONENT** (waarmee een specifieke component geselecteerd wordt), maar ook met het commando **NEXT_COMPONENT**. In beide gevallen **moet** vooraf het component-type geselecteerd zijn.
NEXT_COMPONENT selecteert bij elke aanroep de volgende component van het geselecteerde type.

Het selecteren van componenten in SDWrite is als volgt voor te stellen:
Indien in SDWrite een component wordt geselecteerd, dan wordt bij de betreffende componentnaam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_COMPONENT** op welke component dat commando betrekking heeft.
Het commando **NEXT_COMPONENT** zorgt ervoor dat de markering één naam verder wordt geplaatst.
Indien er nog geen component is geselecteerd, wordt door **NEXT_COMPONENT** de eerste component van het betreffende type geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van componenten van het geselecteerde type worden afgewerkt.

Voorbeeld:
Het volgende script schrijft alle namen van componenten van het type Functie:

```
select_component_type Functie
---->                          (* X *)
while next_component           (* a *)
    BEGIN
    print_component            (* b *)
```

```
END
```

Zolang er nog een volgende component van het type Functie is (a), wordt de naam van deze component geprint (b). Is er geen volgende component meer (a), dan eindigt het script.
Indien met **SELECT_COMPONENT** op positie (* X *) eerst een bepaalde component van het betreffende type wordt geselecteerd, dan zal het script een overzicht geven van alle Functies **vanaf** die geselecteerde component.

Door het commando **NEXT_COMPONENT** wordt automatisch de standaard SDWrite-variabele $SDWcomponent gevuld. Deze variabele bevat op elk moment de naam van de actieve component.
Het voorbeeld-script zou er derhalve ook als volgt uit kunnen zien:

```
select_component_type Functie
while next_component
    BEGIN
    print $SDWcomponent
    END
```

Zie ook: **PRINT_COMPONENT, SELECT_COMPONENT.**

NEXT_COMPONENT_TYPE

Syntax: NEXT_COMPONENT_TYPE

Functie: selecteert het volgende component-type uit de SE. (SDW)

Werking: Om gegevens over een component-type of over een component van dat type te kunnen benaderen, dient eerst dit component-type geselecteerd te worden. Dit kan met het commando **SELECT_COMPONENT_TYPE** (waarmee een specifiek component-type geselecteerd wordt), maar ook met het keyword **NEXT_COMPONENT_TYPE**.
NEXT_COMPONENT_TYPE selecteert bij elke aanroep het volgende component-type.

Het selecteren van component-typen in SDWrite is als volgt voor te stellen:
Indien in SDWrite een component-type wordt geselecteerd, dan wordt bij de betreffende naam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_COMPONENT_TYPE** op welk component-type dat commando betrekking heeft.
Het commando **NEXT_COMPONENT_TYPE** zorgt ervoor dat de markering steeds één naam verder wordt geplaatst. Daarbij wordt bij de eerste aanroep van **NEXT_COMPONENT_TYPE** het eerste component-type van het current SDW-systeem geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van component-typen van het current systeem worden afgewerkt.

Het volgende script schrijft alle component-typen die voorkomen in het SDW-systeem test:

```
sdwsystem test
while next_component_type       (* a *)
    print_component_type        (* b *)
```

Zolang er nog een volgend component-type is (a), wordt de naam van dit component-type geschreven (b). Is er geen

volgend component-type meer (a), dan eindigt het script.

Door **NEXT_COMPONENT_TYPE** wordt automatisch de standaard SDWrite-variabele $SDWcomponent_type gevuld. Deze variabele bevat op elk moment de naam van het actieve component-type.
Het voorbeeld-script zou er derhalve ook als volgt uit kunnen zien:

```
sdwsystem test
while next_component_type
    print $SDWcomponent_type
```

Zie ook: **PRINT_COMPONENT_TYPE, SELECT_COMPONENT_TYPE.**

NEXT_COMPOSITE

Syntax: NEXT_COMPOSITE

Functie: selecteert het volgende samenstellende deel van de geselecteerde samengestelde component. (SDW)

Werking: Om een van de samenstellende delen van een samengestelde component (bijvoorbeeld een Buffer of een Stroom) te benaderen, wordt gebruik gemaakt van een apart begrip: de composite.
Een samenstellingsrubriek wordt hierdoor onderscheiden van de andere rubriekstypen, die uitsluitend als item of generated_item benaderd kunnen worden.

Een voorbeeld: De Stroom 'stroom' is samengesteld uit de substromen 'deelstroom1' en 'deelstroom2'. Om deze samenstelling te benaderen, dienen allereerst het component-type en de component geselecteerd te worden.
Vervolgens kan met:

```
select_item samenstelling
```

en de reguliere SDWrite-rubrieks-commando's de samenstellingsrubriek van de stroom worden benaderd.
Bijvoorbeeld:

```
select_component_type Stroom
select_component stroom
select_item samenstelling
while next_item_text
    print_item_text
```

Indien de samenstellingsrubriek van het type DeMarco is, kan de uitvoer er als volgt uitzien:

```
stroom = deelstroom1 + deelstroom2
```

Is de samenstelling van het type SDW, dan ziet de uitvoer er als volgt uit:

```
deelstroom1          Stroom
deelstroom2          Stroom
```

Het is echter ook mogelijk om met het commando **NEXT_COMPOSITE** (in combinatie met **WHILE**) uitsluitend de namen van de samenstellende delen te selecteren en te schrijven.

Een simpel script hiervoor ziet er als volgt uit:

```
select_component_type Stroom
select_component stroom
while next_composite
    print_composite
```

Dit script heeft als uitvoer:

```
deelstroom1
deelstroom2
```

Het is natuurlijk ook mogelijk voor alle samengestelde componenten uit het current SDW-systeem de samenstellende delen te schrijven. In dat geval dient voor alle componenten van alle component-typen te worden bekeken of ze zijn samengesteld. Een eenvoudig script daarvoor ziet er als volgt uit:

```
while next_component_type              (* a *)
    while next_component               (* b *)
        BEGIN
        if next_composite              (* c *)
            BEGIN
            newline off
            print_component_type       (* d *)
            spaces 1
            print '
            print_component            (* e *)
            newline on
            print ':
            spaces 5
            print_composite            (* f *)
            while next_composite       (* g *)
                BEGIN
                spaces 5
                print_composite        (* h *)
                END
```

```
        newline
        END
    END
```

In dit script wordt van elk component-type (a) elke component bekeken (b), waarbij het component-type (d) en de componentnaam (e) naar het current uitvoerkanaal worden geschreven indien er tenminste één samenstellend deel is (c). In dat geval wordt op een nieuwe regel de naam van dat samenstellende deel geschreven (f) en worden bovendien alle overige samenstellende delen weggeschreven (g, h).

De uitvoer van dit script kan er als volgt uitzien:

```
Stroom 'stroom':
    deelstroom1
    deelstroom2

Stroom 'invoer':
    deel_invoer1
    deel_invoer2
    deel_invoer3
```

Door **NEXT_COMPOSITE** wordt automatisch de standaard SDWrite-variabele $SDWcomposite gevuld. Deze variabele bevat op elk moment de naam van het actieve samenstellende deel. Het afdrukken van de samenstelling van een component zou er derhalve ook als volgt uit kunnen zien:

```
select_component_type Stroom
select_component stroom
while next_composite
    print $SDWcomposite
```

Zie ook: **PRINT_COMPOSITE, NEXT_ITEM, SELECT_ITEM.**

NEXT_DIAGRAM

Syntax: NEXT_DIAGRAM

Functie: selecteert het volgende diagram van het geselecteerde diagram-type uit de SE. (SDW)

Werking: Om gegevens over een diagram te kunnen benaderen, dient dit diagram eerst geselecteerd te worden. Dit kan met het commando **SELECT_DIAGRAM** (waarmee een specifiek diagram geselecteerd wordt), maar ook met **NEXT_DIAGRAM**. In beide gevallen **moet** vooraf het diagram-type geselecteerd zijn.
NEXT_DIAGRAM selecteert bij elke aanroep de volgende component van het geselecteerde type.

Het selecteren van diagrammen in SDWrite is als volgt voor te stellen:
Indien in SDWrite een diagram wordt geselecteerd, dan wordt bij de betreffende diagramnaam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_DIAGRAM** op welk diagram dat commando betrekking heeft.
Het commando **NEXT_DIAGRAM** zorgt ervoor dat de markering één naam verder wordt geplaatst.
Indien er nog geen diagram is geselecteerd, wordt door **NEXT_DIAGRAM** het eerste diagram van het betreffende type geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van diagrammen van het geselecteerde type worden afgewerkt.

Voorbeeld:
Het volgende script schrijft alle namen van diagrammen van het type Data Flow Diagram:

```
select_diagram_type Data Flow Diagram
---->                          (* X *)
while next_diagram             (* a *)
    BEGIN
    print_diagram_name         (* b *)
    END
```

Zolang er nog een volgend diagram van het type Data Flow Diagram is (a), wordt de naam van dit diagram geprint (b). Is er geen volgend diagram meer (a), dan eindigt het script.
Indien met **SELECT_DIAGRAM** op positie (* X *) eerst een bepaald diagram van het betreffende type wordt geselecteerd, dan zal het script een overzicht geven van alle Data Flow Diagrams **vanaf** dat geselecteerde diagram.

Door het commando **NEXT_DIAGRAM** wordt automatisch de standaard SDWrite-variabele $SDWdiagram gevuld. Deze variabele bevat op elk moment de naam van het actieve diagram.
Het voorbeeld-script zou er derhalve ook als volgt uit kunnen zien:

```
select_component_type Data Flow Diagram
while next_diagram
    BEGIN
    print $SDWdiagram
    END
```

Zie ook: **PRINT_DIAGRAM, PRINT_DIAGRAM_NAME, SELECT_DIAGRAM.**

NEXT_DIAGRAM_TYPE

Syntax: NEXT_DIAGRAM_TYPE

Functie: selecteert het volgende diagram-type uit de SE. (SDW)

Werking: Om de gegevens over een diagram-type of over een diagram van dat type te kunnen benaderen, dient dit diagram-type eerst geselecteerd te worden. Dit kan met het commando **SELECT_DIAGRAM_TYPE** (waarmee een specifiek component-type geselecteerd wordt), maar ook met **NEXT_DIAGRAM_TYPE**. **NEXT_DIAGRAM_TYPE** selecteert bij elke aanroep het volgende diagram-type.

Het selecteren van diagram-typen in SDWrite is als volgt voor te stellen:
Indien in SDWrite een diagram-type wordt geselecteerd, dan wordt bij de betreffende naam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_DIAGRAM_TYPE** op welk diagram-type dat commando betrekking heeft.
Het commando **NEXT_DIAGRAM_TYPE** zorgt ervoor dat de markering steeds één naam verder wordt geplaatst. Daarbij wordt bij de eerste aanroep van **NEXT_DIAGRAM_TYPE** het eerste diagram-type van het actieve SDW-systeem geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van diagram-typen van het current systeem worden afgewerkt.

Het volgende script schrijft alle diagram-typen die voorkomen in het SDW-systeem test:

```
sdwsystem test
while next_diagram_type      (* a *)
    print_diagram_type       (* b *)
```

Zolang er nog een volgend diagram-type is (a), wordt de naam van dit diagram-type geschreven (b). Is er geen volgend diagram-type meer (a), dan eindigt het script.

Door **NEXT_DIAGRAM_TYPE** wordt automatisch de standaard SDWrite-variabele $SDWdiagram_type gevuld. Deze variabele bevat op elk moment de naam van het actieve diagram-type. Het voorbeeld-script zou er derhalve ook als volgt uit kunnen zien:

```
sdwsystem test
while next_diagram_type
    print $SDWdiagram_type
```

Zie ook: **SELECT_DIAGRAM_TYPE, PRINT_DIAGRAM_TYPE.**

NEXT_GENERATED_ITEM

Syntax: NEXT_GENERATED_ITEM

Functie: selecteert de volgende gegenereerde rubriek van het geselecteerde component-type. (SDW)

Werking: Om de inhoud van een gegenereerde rubriek in de Systeem Encyclopedie te kunnen benaderen, dient deze rubriek eerst geselecteerd te worden. Dit kan op twee manieren: met **SELECT_GENERATED_ITEM** (waarmee een specifieke gegenereerde rubriek geselecteerd wordt), maar ook met **NEXT_GENERATED_ITEM**. Dit laatste commando selecteert bij elke aanroep de volgende gegenereerde rubriek van het geselecteerde component-type.
In beide gevallen **moet** vooraf het component-type geselecteerd zijn. Of vervolgens eerst de component en dan de rubriek, of eerst de rubriek en dan de component geselecteerd wordt, is niet van belang.
Om de inhoud van een rubriek te benaderen (met **NEXT_GENERATED_ITEM_TEXT**) dienen echter zowel de rubriek als de component geselecteerd te zijn.

Het selecteren van rubrieken in SDWrite is als volgt voor te stellen:
Indien in SDWrite een gegenereerde rubriek wordt geselecteerd, dan wordt bij de betreffende rubrieksnaam als het ware een markering geplaatst.
Dankzij die markering weet SDWrite bij commando's als **PRINT_GENERATED_ITEM_NAME** op welke rubriek dat commando betrekking heeft.
Het commando **NEXT_GENERATED_ITEM** zorgt ervoor dat de markering steeds één naam verder wordt geplaatst.
Bij de eerste aanroep van het commando wordt de eerste gegenereerde rubriek van het betreffende component-type of de betreffende component geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van gegenereerde rubrieken van het geselecteerde component-type worden afgewerkt.

Het volgende voorbeeldscript schrijft alle namen van gegenereerde rubrieken van het component-type Functie:

```
select_component_type Functie
while next_generated_item          (* a *)
    print_generated_item_name      (* b *)
```

Zolang er nog een volgende gegenereerde rubriek van het type Functie is (a), wordt de naam van deze rubriek geprint (b). Is er geen volgende rubriek meer (a), dan eindigt het script.

Door **NEXT_GENERATED_ITEM** wordt automatisch de standaard SDWrite-variabele $SDWgenerated_item gevuld. Deze variabele bevat op elk moment de naam van de actieve gegenereerde rubriek.
Het voorbeeld-script zou er derhalve ook als volgt uit kunnen zien:

```
select_component_type Functie
while next_generated_item
    print $SDWgenerated_item
```

Zie ook: **SELECT_GENERATED_ITEM, PRINT_GENERATED_ITEM_TEXT, PRINT_GENERATED_ITEM_HEADER, NEXT_COMPOSITE, NEXT_ITEM, PRINT_GENERATED_ITEM_NAME.**

NEXT_GENERATED_ITEM_TEXT

Syntax: NEXT_GENERATED_ITEM_TEXT

Functie: selecteert de volgende tekstregel van de geselecteerde gegenereerde rubriek. (SDW)

Werking: Met SDWrite kan de inhoud van een gegenereerde rubriek alleen **regel voor regel** worden verwerkt en weggeschreven naar het current uitvoerkanaal of een variabele. Om een regel te kunnen wegschrijven (met **PRINT_GENERATED_ITEM_TEXT**), dient de betreffende regel echter eerst met **NEXT_GENERATED_ITEM_TEXT** geselecteerd te worden.
Voorafgaand aan dat commando **moeten** eerst het component-type, de component èn de gegenereerde rubriek geselecteerd te zijn. Er zijn slechts twee volgordes toegestaan:

1 selecteren van het component-type
2 selecteren van de component
3 selecteren van de rubriek
4 **NEXT_GENERATED_ITEM_TEXT**
5 **PRINT_GENERATED_ITEM_TEXT**

en

1 selecteren van het component-type
2 selecteren van de rubriek
3 selecteren van de component
4 **NEXT_GENERATED_ITEM_TEXT**
5 **PRINT_GENERATED_ITEM_TEXT**

De eerste keer dat het **NEXT_GENERATED_ITEM_TEXT**-commando wordt gegeven, wordt de eerste regel geselecteerd van de betreffende rubriek.
Bij elke volgende aanroep wordt de volgende regel van de rubriek geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele inhoud van een rubriek worden benaderd.

Voorbeeld:
Het volgende script schrijft de inhoud van de gegenereerde rubriek 'invoerstromen' van de Functie 'invoeren' naar het current uitvoerkanaal.

```
select_component_type Functie
select_component invoeren
select_generated_item invoerstromen

while next_generated_item_text
    print_generated_item_text
```

Tijdens het verwerken van een gegenereerde rubriek worden (afhankelijk van de betreffende rubriek) een of meer standaard SDWrite-variabelen gevuld.

Zie ook: **SELECT_GENERATED_ITEM, PRINT_GENERATED_ITEM_TEXT, PRINT_GENERATED_ITEM_HEADER, NEXT_GENERATED_ITEM, PRINT_GENERATED_ITEM_NAME.**

SDWrite-variabelen
Module-handleidingen: Appendix (Gegenereerde rubrieken)

NEXT_ITEM

Syntax: NEXT_ITEM

Functie: selecteert de volgende vrij definieerbare rubriek van het geselecteerde component-type. (SDW)

Werking: Om de inhoud van een vrije definieerbare rubriek in de Systeem Encyclopedie te kunnen benaderen, dient deze rubriek eerst geselecteerd te worden. Dit kan op twee manieren: met het commando **SELECT_ITEM** (waarmee een specifieke rubriek geselecteerd wordt), maar ook met **NEXT_ITEM**. Dit laatste commando selecteert bij elke aanroep de volgende vrij definieerbare rubriek van het geselecteerde component-type.
In beide gevallen **moet** vooraf het component-type geselecteerd zijn. Of vervolgens eerst de component en dan de rubriek, of eerst de rubriek en dan de component geselecteerd wordt, is niet van belang.
Om de inhoud van een rubriek te benaderen (met **NEXT_ITEM_TEXT**) dienen echter zowel de rubriek als de component geselecteerd te zijn.

N.B. Voor rubrieken van het type 'samenstelling' gelden andere commando's. Zie hiervoor **NEXT_COMPOSITE** en **PRINT_COMPOSITE**.

Het selecteren van rubrieken in SDWrite is als volgt voor te stellen:
Indien in SDWrite een vrij definieerbare rubriek wordt geselecteerd, dan wordt bij de betreffende rubrieksnaam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij op welke rubriek commando's als **PRINT_ITEM_NAME**-commando betrekking hebben.
Het commando **NEXT_ITEM** zorgt ervoor dat de markering steeds één naam verder wordt geplaatst. Daarbij wordt bij de eerste aanroep van **NEXT_ITEM** de eerste rubriek van het betreffende component-type of de betreffende component geselecteerd. Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele lijst van vrij definieerbare rubrieken van het geselecteerde component-type

worden afgewerkt.

Voorbeeld:
Het volgende script schrijft alle namen van vrij definieerbare rubrieken van het component-type Functie:

```
select_component_type Functie
while next_item                 (* a *)
    print_item_name             (* b *)
```

Zolang er nog een volgende gegenereerde rubriek van het type Functie is (a), wordt de naam van deze rubriek geprint (b). Is er geen volgende rubriek meer (a), dan eindigt het script.

Door **NEXT_ITEM** wordt automatisch de standaard SDWrite-variabele $SDWitem gevuld. Deze variabele bevat op elk moment de naam van de actieve rubriek.
Het voorbeeld-script zou er derhalve ook als volgt uit kunnen zien:

```
select_component_type Functie
while next_item
    print $SDWitem
```

Zie ook: **NEXT_COMPOSITE, SELECT_ITEM, NEXT_GENERATED_ITEM, PRINT_ITEM_HEADER, PRINT_ITEM_NAME, PRINT_ITEM_TEXT.**

ook voor associatie
uitvoer print_item_text
text associatie type

NEXT_ITEM_TEXT

Syntax: NEXT_ITEM_TEXT

Functie: selecteert de volgende tekstregel van de geselecteerde rubriek. (SDW)

Werking: Met SDWrite kan de inhoud van vrij definieerbare rubriek alleen **regel voor regel** worden verwerkt en weggeschreven naar het current uitvoerkanaal of een variabele. Om een regel te kunnen wegschrijven (met **PRINT_ITEM_TEXT**), dient de betreffende regel echter eerst geselecteerd te zijn met **NEXT_ITEM_TEXT**.
Voorafgaand aan het dat commando **moeten** eerst het component-type, de component èn de rubriek geselecteerd te zijn.
Er zijn slechts twee volgordes toegestaan:

1. selecteren van het component-type
2. selecteren van de component
3. selecteren van de rubriek
4. **NEXT_ITEM_TEXT**
5. **PRINT_ITEM_TEXT**

en

1. selecteren van het component-type
2. selecteren van de rubriek
3. selecteren van de component
4. **NEXT_ITEM_TEXT**
5. **PRINT_ITEM_TEXT**

De eerste keer dat het commando **NEXT_ITEM_TEXT** wordt gegeven, wordt de eerste regel geselecteerd van de betreffende rubriek. Bij elke volgende aanroep wordt de volgende regel van de rubriek geselecteerd.
Door herhaalde aanroep van het commando (bijvoorbeeld in een **WHILE**-constructie) kan de hele inhoud van een rubriek worden benaderd.

Voorbeeld:
Het volgende script schrijft de inhoud van de rubriek 'omschrijving' van de Functie 'invoeren' naar het current uitvoerkanaal.

```
select_component_type Functie
select_component invoeren
select_item omschrijving

while next_item_text
    print_item_text
```

Tijdens het verwerken van rubrieken van het type Associatie of Samenstelling volgens SDW worden per regel automatisch de standaard SDWrite-variabelen $SDWitem_component en $SDWitem_component_type gevuld.

Zie ook: **SELECT_ITEM, NEXT_GENERATED_ITEM_TEXT, NEXT_ITEM, PRINT_ITEM_HEADER, PRINT_ITEM_NAME, PRINT_ITEM_TEXT.**

NOT

Syntax: <IF | WHILE> NOT <voorwaarde> <actie>

Functie: bewerkstelligt dat de gegeven actie wordt uitgevoerd indien **niet** aan de gegeven voorwaarde wordt voldaan. (PRG)

Werking: Het keyword **NOT** is alleen van belang in combinatie met **WHILE** of **IF**. In een voorwaardelijke constructie van de vorm:

if <voorwaarde> dan <actie>

of

while <voorwaarde> dan <actie>

wordt de opgegeven actie (één commando of een reeks commando's tussen **BEGIN** en **END**) uitgevoerd indien aan de gegeven voorwaarde wordt voldaan.
Door toevoeging van **NOT** tussen **IF** / **WHILE** en de voorwaarde, wordt ervoor gezorgd dat de actie alleen wordt uitgevoerd indien aan het tegendeel van de gegeven voorwaarde wordt voldaan.

Een voorbeeld:
De volgende constructie leidt ertoe dat iemand niet toegelaten wordt indien het opgegeven antwoord 'paswoord' is:

```
display Toets Uw paswoord in >
set $antwoord prompt
if equal $antwoord paswoord
    print Niet toegelaten
```

Het kan echter wenselijker worden geacht, om de gebruiker minder kansen te geven om door te gaan. Tenslotte is er doorgaans niet slechts één fout paswoord, maar slechts één góede.

Het voorbeeld kan daarvoor met **NOT** zodanig worden aangepast, dat alleen iemand die 'XScode' intoetst, wordt

toegelaten:

```
display Toets Uw paswoord in >
set $antwoord prompt
if NOT equal $antwoord XScode
    print Niet toegelaten
```

Zie ook: **BEGIN, END, IF, WHILE.**

OUTPUT

Syntax: OUTPUT [$variabele | uitvoerkanaal] [SDWDRIVER]

Functie: sluit het current uitvoerkanaal en opent het gegeven nieuwe uitvoerkanaal. Indien geen parameters worden meegegeven, wordt het default-uitvoerkanaal (het beeldscherm) geopend en wordt de uitvoer zonder printer-besturing geschreven. (IO)

Werking: Default schrijft SDWrite alle uitvoer naar het beeldscherm. Het is echter ook mogelijk te schrijven naar andere uitvoerkanalen, zoals een printer of een file.
Bovendien kan worden aangegeven of bij de uitvoer gebruik gemaakt dient te worden van de geselecteerde SDW-printerdriver of dat de uitvoer als 'platte' ASCII-file geschreven wordt.

Indien wordt gekozen voor aansturing via de printerdriver, wordt standaard gebruik gemaakt van de printerdriver die is geselecteerd bij het installeren van het werkstation (sdw_drv.mpt in de LIBWS-directory). Indien met **SELECT_SDWDRIVER** een andere driver is geselecteerd, wordt die driver gebruikt.

Om uitvoer naar een printer te sturen, kan bijvoorbeeld een van de volgende commando's gegeven te worden:

```
output prn
output lpt1
```

Om aan te geven dat hierbij gebruik gemaakt moet worden van de voor SDW ingestelde printerdriver, dient als tweede argument 'SDWDRIVER' te worden meegegeven:

```
output prn SDWDRIVER
output lpt1 SDWDRIVER
```

In alle gevallen dient het commando gegeven te worden **voorafgaand** aan de schrijfopdrachten die naar de printer gestuurd moeten worden.

Bij het selecteren van een file als current uitvoerkanaal dient rekening gehouden te worden met aanduidingen voor drive en pad. Afhankelijk daarvan zou de uitvoer naar een en dezelfde file gestuurd kunnen worden met elk van de volgende opdrachten:

```
output uitvoer.txt
output \data\uitvoer.txt
output c:\data\uitvoer.txt
etc.
```

Om in de loop van een script de uitvoer (al dan niet tijdelijk) weer naar het beeldscherm te laten sturen, dient opnieuw het **OUTPUT**-commando gegeven te worden, dit maal zónder parameter:

```
output
```

Het is mogelijk in één script meerdere malen het actieve uitvoerkanaal te veranderen.
Voorbeeld:

```
output prn
print Dit wordt gestuurd naar de printer
output file.f
print Dit komt in de file 'file.f'
print En dit ook.
output
print Dit gaat naar het beeldscherm
```

Overigens is het niet altijd nodig om **OUTPUT** (zonder parameters) te gebruiken in een script dat ook (en vooral) schrijft naar een file of de printer. In dat geval kan ook vaak het **DISPLAY**-commando gebruikt worden.
Te denken valt hierbij aan echo's naar het scherm die de gebruiker op de hoogte houden van het verloop van een script dat anders ongezien zou voorbijgaan (omdat alle uitvoer bijvoorbeeld naar een file gaat).

Het is ook mogelijk de uitvoerbestemming in de vorm van een **variabele** aan **OUTPUT** mee te geven. Dit is vooral handig in interactieve scripts. Bijvoorbeeld:

```
set $out prompt Naar welke file?
output $out
```

Als uitvoerkanaal wordt nu de filenaam (of de printer) gekozen, die door de gebruiker wordt ingetoetst.
De variabele die aan het **OUTPUT**-commando wordt meegegeven, dient vooraf met **SET** geïnitialiseerd te zijn. Is dat niet gebeurd, dan volgt een foutmelding en wordt de verwerking van het script afgebroken.

Indien een diagram wordt afgedrukt, wordt **automatisch** gebruik gemaakt van de actieve printerdriver. Het argument SDWDRIVER is in dat geval niet nodig.

Zie ook: **DISPLAY, LINE_OPEN, SELECT_SDWDRIVER.**

PAGELENGTH

Syntax: PAGELENGTH <aantal>

Functie: zorgt ervoor dat na het opgegeven aantal regels de header wordt geprint. (Lay Out)

Werking: Met het **PAGELENGTH**-commando wordt het aantal regels opgegeven dat op een pagina past. Default is dit 24.
Daarbij wordt niet uitgegaan van een werkelijke pagina (een A4 of een A3), maar van de pagina zoals de gebruiker die wil zien. Het is derhalve niet noodzakelijk voor 11"-papier een paginalengte van 66 op te geven, enzovoorts.

Een voorbeeld:
Indien voor speciale doeleinden drie SDWrite-pagina's op een A4 zouden moeten passen, kan dit bereikt worden door het commando:

```
pagelength 22
```

Dit heeft tot gevolg dat na 22 geschreven regels de **HEADER** wordt geprint. Of er na 22 regels ook een formfeed wordt gegeven, hangt volledig af van de opdrachten in de **HEADER**. Indien in deze **HEADER** géén formfeed wordt gegenereerd, zal het overschrijden van de SDWrite-pagina-lengte ook **geen** formfeed tot gevolg hebben.

Na het aantal met **PAGELENGTH** opgegeven regels wordt dus **niet automatisch een formfeed** gegenereerd !!!
Het enige gevolg van een overschrijding van het opgegeven aantal regels is dat de **HEADER**-statements worden uitgevoerd.

Houdt er rekening mee dat bij het schrijven naar een printer ook formfeeds kunnen voorkomen, die niet door het script worden gegenereerd. Veelal genereren printers na een bepaald aantal regels automatisch een formfeed, zelfs als **FORMFEED** in SDWrite OFF staat; zie hiervoor de printer-handleiding.

Zie ook: **EJECT, FORMFEED, HEADER.**

PAGENO

Syntax: PAGENO [start]

Functie: schrijft het bladzijdenummer, en verhoogt het bij elke aanroep van **EJECT** met 1. (Lay Out)

Werking: Om een rapport te voorzien van pagina-nummering, dient het commando **PAGENO** gebruikt te worden. Dit commando schrijft het pagina-nummer naar het current uitvoerkanaal.
Het pagina-nummer wordt intern door SDW bijgehouden en bij elk voorkomen van **EJECT** met 1 (één) verhoogd. Standaard begint SDW te tellen bij 1, tenzij een ander nummer als argument is opgegeven.

De eerste maal dat het commando **PAGENO** wordt gegeven, wordt standaard als pagina-nummer 1 geschreven, na **EJECT** 2, na wederom **EJECT** 3, etc.
Het (onzin)script:

```
pageno
eject
pageno
pageno
eject
eject
pageno
```

zal derhalve als uitvoer hebben:

```
1
2
2
4
```

De meest aangewezen plaats voor het **PAGENO**-commando is de **HEADER**.
Voorbeeld:

```
HEADER
BEGIN
    eject
```

```
    formfeed on
    print ------------------------------
    Print SDWrite-rapport no.2
    print Systeem ;
    column 16
    print : ;
    spaces 2
    sdwsystem
    print Datum ;
    column 16
    print : ;
    spaces 2
    date
    print Pagina ;
    column 16
    print : ;
    spaces 2
    pageno
    print ------------------------------
END
```

met uitvoer in de volgende vorm:

```
------------------------------
SDWrite-rapport no. 2
Systeem     :   data
Datum       :   10-04-1993
Pagina      :   13
------------------------------
```

Zie ook: **HEADER, PAGELENGTH.**

PRINT

Syntax: PRINT {$varnaam | \tekst | tekst }

Functie: schrijft de gegeven combinatie van variabelen en/of tekst naar het current uitvoerkanaal of naar een variabele. (IO)

Werking: Om tekst en/of waarden van variabelen naar het current uitvoerkanaal te schrijven, heeft SDWrite een wijd scala aan **PRINT**-opdrachten (print, print_header, print_component, print_number, etc.).
Al deze **PRINT**-opdrachten kunnen bovendien gebruikt worden om een variabele van een waarde te voorzien.
De mogelijkheden met **PRINT** zijn daardoor bijna onuitputtelijk.

De simpelste vorm van een **PRINT**-opdracht is de volgende:

```
print SDWrite-rapport
```

waardoor de tekst 'SDWrite-rapport' naar het current uitvoerkanaal wordt geschreven.
Of daarbij een newline wordt gegeven hangt af van twee instellingen:

- de instelling van **NEWLINE**: indien **NEWLINE** OFF staat wordt na een **PRINT**-opdracht geen newline gegeven; in het geval dat **NEWLINE** ON staat wel

- de instelling van **SEMICOLON**: indien een **PRINT**-opdracht eindigt op een puntkomma (;) betekent dat standaard dat géén newline moet worden geschreven. Door **SEMICOLON** te gebruiken, kan echter ook aangegeven worden dat een afsluitende puntkomma gewoon als een teken in de te schrijven tekst moet worden beschouwd. In dat geval bepaalt de instelling van **NEWLINE** of er een newline wordt gegeven of niet

Dit houdt onder andere in dat afhankelijk van de instelling van **NEWLINE** door het commando:

```
print          (* zonder argument *)
```

een newline wordt geschreven of niets.

Het is mogelijk deze eenvoudige **PRINT**-opdracht iets ingewikkelder te maken door er een of meer variabelen in op te nemen:

```
set $naam SDWrite-
set $aard voorbeeld
print Dit is een $aard van $naam
```

met als resultaat:

```
Dit is een voorbeeld van SDWrite-
```

De spaties tussen de argumenten worden ook geschreven !!!

Dit betekent dat de **PRINT**-opdrachten:

```
print $naam $aard
```

en

```
print $naam$aard
```

respectievelijk als uitvoer hebben:

```
SDWrite- voorbeeld
```

en

```
SDWrite-voorbeeld
```

In combinatie met het **SET**-commando kunnen **PRINT**-opdrachten ook gebruikt worden om een waarde toe te kennen aan een **variabele**:

```
set $var print tekst
```

leidt ertoe dat de variabele $var de waarde 'tekst' krijgt.
Met deze constructie kunnen ook strings geconcateneerd worden:

```
set $nieuw print $naam$aard
```

met als waarde voor $nieuw 'SDWrite-voorbeeld'.

Om in SDWrite-variabelen spaties op te nemen is deze vorm van toekenning (via een samengestelde **PRINT**-opdracht) zelfs vereist. Directe toekenning van een waarde met een spatie, zoals:

```
set $nieuw SDWrite- voorbeeld
```

leidt er namelijk toe dat alleen de tekst vóór de spatie als waarde aan de variabele wordt toegekend. In dit geval zal $nieuw dus de waarde 'SDWrite-' krijgen.
Wordt echter gebruik gemaakt van de opdracht:

```
set $nieuw print $naam $aard
```

dan zal wèl de waarde 'SDWrite- voorbeeld' aan $nieuw worden toegekend.

Door gebruik te maken van het scheidingsteken '\' kunnen variabelen en vaste teksten direct achter elkaar geschreven worden.
Uitgaande van het voorbeeld levert de opdracht:

```
print $naam-voorbeeld
```

een foutmelding op, aangezien de variabele '$naam-voorbeeld' niet bestaat. Om toch op deze wijze de tekst 'SDWrite-voorbeeld' te kunnen genereren, dient het scheidingsteken gebruikt te worden:

```
print $naam\-voorbeeld
```

In dit geval wordt de variabele $naam geschreven, direct gevolgd door de tekst '-voorbeeld'.

Indien het gewenst is, dat bij het schrijven van een variabele niet de waarde, maar de variabele zelf wordt geschreven, dient gebruik gemaakt te worden van het keyword **DOLLAR**.
Het volgende script:

```
set $tekst print Voorbeeld
print $tekst
dollar off
print $tekst
```

zal als uitvoer opleveren:

```
Voorbeeld
$tekst
```

Zie ook: **DOLLAR, SET, NEWLINE, SEMICOLON.**

PRINT_ADDITIONAL_DIAGRAM

Syntax: PRINT_ADDITIONAL_DIAGRAM

Functie: schrijft eventuele vervolgbladen van een diagram naar het current uitvoerkanaal. (SDW)

Werking: Een aantal diagram-technieken maakt gebruik van zogenaamde vervolgbladen. Hierop staat aanvullende informatie over het diagram. Voorbeelden zijn het Instructie-schema bij Detail-processchema's en de State Transition Tabel bij een State Transition Diagram.

Met het commando **PRINT_ADDITIONAL_DIAGRAM** kunnen de eventuele vervolgbladen van het geselecteerde diagram naar het geselecteerde uitvoerkanaal geschreven worden. Voorafgaand dient het diagram geselecteerd te zijn (met **SELECT_DIAGRAM** of **NEXT_DIAGRAM**). Is dat nog niet gebeurd, dan volgt een foutmelding.
De werking van het commando is als volgt voor te stellen:
Als een diagram wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_ADDITIONAL_DIAGRAM**-commando wordt gegeven, wordt gekeken bij welk diagram deze markering staat, en wordt vervolgens het diagram afgedrukt. Door een ander diagram te selecteren, verschuift de markering, waardoor het **PRINT_ADDITIONAL_DIAGRAM**-commando een ander diagram zal schrijven.

Bij het afdrukken van een diagram en/of de vervolgbladen dient voorafgaand altijd het uitvoerkanaal geselecteerd zijn. Hierbij wordt **automatisch** gebruik gemaakt van de geselecteerde printerdriver, zodat het argument SDWDRIVER niet nodig is:

```
output uitvoer.dia
print_additional_diagram uitvoer
```

Indien een diagram meer dan een vervolgblad heeft, worden automatisch **alle** vervolgbladen van dat diagram afgedrukt.

Zie ook: **SELECT_DIAGRAM_TYPE, SELECT_DIAGRAM, PRINT_DIAGRAM.**

PRINT_COMPONENT

Syntax: PRINT_COMPONENT

Functie: schrijft de geselecteerde componentnaam naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het commando **PRINT_COMPONENT** wordt de current component geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de component eerst geselecteerd te zijn (met **SELECT_COMPONENT** of **NEXT_COMPONENT**). Is dat nog niet gebeurd, dan volgt een foutmelding.
De werking van het commando is als volgt voor te stellen:
Als een component wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_COMPONENT**-commando wordt gegeven, wordt gekeken bij welke component deze markering staat, en wordt vervolgens de naam van deze component geschreven. Door een andere component te selecteren, verschuift de markering, waardoor het **PRINT_COMPONENT**-commando een andere componentnaam zal schrijven.

Via het **SET**-commando kan een componentnaam ook als waarde aan een variabele worden toegekend. Het script:

```
select_component_type Functie
set $aantal 0
while next_component
    BEGIN
    add $aantal 1
    set $comp print_component
    print $aantal - $comp
    END
```

zal alle functie-namen naar het current uitvoerkanaal schrijven, met een nummer ervoor, bijvoorbeeld:

```
1 - uitvoeren
2 - invoeren
```

Aangezien de naam van de actieve component door SDWrite automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWcomponent, kan in plaats van het commando **PRINT_COMPONENT** ook altijd gebruik gemaakt worden van:

```
print $SDWcomponent
```

Zie ook: **NEXT_COMPONENT, SELECT_COMPONENT, SET.**

PRINT_COMPONENT_TYPE

Syntax: PRINT_COMPONENT_TYPE

Functie: schrijft het geselecteerde component-type naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het commando **PRINT_COMPONENT_TYPE** wordt het current component-type geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient het component-type eerst geselecteerd te zijn (dit kan met **SELECT_COMPONENT_TYPE** of **NEXT_COMPONENT_TYPE**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een component-type wordt geselecteerd, wordt dit door SDWrite als het ware van een markering voorzien. Indien het **PRINT_COMPONENT_TYPE**-commando wordt gegeven, wordt gekeken bij welk component-type deze markering staat, en wordt vervolgens de naam van dit component-type geschreven. Door een ander component-type te selecteren, verschuift de markering, waardoor het commando **PRINT_COMPONENT_TYPE** een ander component-type zal schrijven.

In combinatie met het **SET**-commando kan de naam van een component-type ook als waarde aan een variabele worden toegekend.

Het script:

```
sdwsystem data
set $aantal 0
while next_component_type
    BEGIN
    add $aantal 1
    set $type print_component_type
    print $aantal - $type
    END
```

zal alle component-typen van het SDW-systeem 'data' naar het current uitvoerkanaal schrijven, met een nummer ervoor.

De uitvoer kan er als volgt uitzien:

```
1 - Buffer
2 - Functie
3 - Gegevenselement
4 - Stroom
<etc.>
```

Aangezien de naam van het actieve component-type door SDWrite automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWcomponent_type, kan in plaats van het commando **PRINT_COMPONENT_TYPE** ook altijd gebruik gemaakt worden van:

```
print $SDWcomponent_type
```

Zie ook: **NEXT_COMPONENT_TYPE, SELECT_COMPONENT_TYPE, SET.**

PRINT_COMPOSITE

Syntax: PRINT_COMPOSITE

Functie: schrijft de naam van de geselecteerde samenstellende component naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het commando **PRINT_COMPOSITE** wordt het geselecteerde deel van een samenstelling geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de composite eerst geselecteerd te zijn (met **NEXT_COMPOSITE**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een composite wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_COMPOSITE**-commando wordt gegeven, wordt gekeken bij welke composite deze markering staat, en wordt vervolgens de naam van deze composite geschreven. Door een andere composite te selecteren, verschuift de markering, waardoor het commando **PRINT_COMPOSITE** een andere composite-naam zal schrijven.

In combinatie met het **SET**-commando kan de naam van een composite ook als waarde aan een variabele worden toegekend.

Het script:

```
select_component_type Stroom
select_component invoerstroom
set $aantal 0
while next_composite
    BEGIN
    add $aantal 1
    set $comp print_composite
    print $aantal - $comp
    END
```

zal alle samenstellende delen van de Stroom 'invoerstroom'

naar het current uitvoerkanaal schrijven, met een nummer ervoor.
De uitvoer kan er als volgt uitzien:

```
1 - deelstroom1
2 - deelstroom2
```

Aangezien de naam van de actieve composite door SDWrite automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWcomposite kan in plaats van het commando **PRINT_COMPOSITE** ook altijd gebruik gemaakt worden van:

```
print $SDWcomposite
```

Bovendien worden bij de verwerking van een Samenstelling van het type SDW per regel de standaard SDWrite-variabelen $SDWitem_component en $SDWitem_component_type gevuld, zodat per regel ook gebruikt gemaakt kan worden van:

```
print $SDWitem_component
```

Zie ook: **NEXT_COMPOSITE, NEXT_ITEM, PRINT_ITEM_HEADER, PRINT_ITEM_NAME, PRINT_ITEM_TEXT, SELECT_ITEM, SET.**

PRINT_COMPOSITE_TYPE

Syntax: PRINT_COMPOSITE_TYPE

Functie: schrijft het type van de geselecteerde samenstellende component naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het commando **PRINT_COMPOSITE_TYPE** wordt het component-type van het geselecteerde deel van een samenstelling geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de composite eerst geselecteerd te zijn (met **NEXT_COMPOSITE**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een composite wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_COMPOSITE_TYPE**-commando wordt gegeven, wordt gekeken bij welke composite deze markering staat, en wordt vervolgens het component-type van deze composite geschreven. Door een andere composite te selecteren, verschuift de markering, waardoor het commando **PRINT_COMPOSITE_TYPE** het type van de volgende composite zal schrijven.

In combinatie met het **SET**-commando kan het type van een composite ook als waarde aan een variabele worden toegekend.

Het script:

```
select_component_type Stroom
select_component invoerstroom
set $aantal 0
while next_composite
    BEGIN
    add $aantal 1
    set $comp print_composite
    set $ctype print_composite_type
    display $ctype $comp
```

```
END
```

zal alle samenstellende delen van de Stroom 'invoerstroom' naar het current uitvoerkanaal schrijven, met het component-type ervoor.
De uitvoer kan er als volgt uitzien:

```
Gegevenselement deelelement1
Gegevenselement deelelement2
Stroom deelstroom1
```

Aangezien bij de verwerking van een Samenstelling van het type SDW per regel de standaard SDWrite-variabelen $SDWitem_component en $SDWitem_component_type worden gevuld, kan voor dat rubriekstype per regel ook gebruikt gemaakt kan worden van:

```
print $SDWitem_component_type
```

Zie ook: **PRINT_COMPOSITE, NEXT_COMPOSITE.**

PRINT_DIAGRAM

Syntax: PRINT_DIAGRAM

Functie: afdrukken van het geselecteerde diagram. (SDW)

Werking: Met het commando **PRINT_DIAGRAM** wordt het geselecteerde diagram geschreven naar het geselecteerde uitvoerkanaal. Voorafgaand dient het diagram geselecteerd te zijn (met **SELECT_DIAGRAM** of **NEXT_DIAGRAM**). Is dat nog niet gebeurd, dan volgt een foutmelding.
De werking van het commando is als volgt voor te stellen:
Als een diagram wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_DIAGRAM**-commando wordt gegeven, wordt gekeken bij welk diagram deze markering staat, en wordt vervolgens het diagram afgedrukt. Door een ander diagram te selecteren, verschuift de markering, waardoor het **PRINT_DIAGRAM**-commando een ander diagram zal schrijven.

★ Bij het afdrukken van een diagram wordt automatisch gebruik gemaakt van de actieve printerdriver.

★ Het afdrukken van diagrammen met **PRINT_DIAGRAM** is uitsluitend mogelijk vanuit de commandline-versie van SDWrite en vanuit SDW-Reporter.

★ Na het commando **PRINT_DIAGRAM** wordt automatisch het commando **EJECT** uitgevoerd.

★ Het is **niet** mogelijk op één pagina een diagram te combineren met tekst.

★ Indien het geselecteerde diagram-type **vervolgbladen** heeft (zoals een Instructie-schema), kunnen de bij een diagram horende vervolgbladen worden geprint met behulp van **PRINT_ADDITIONAL_DIAGRAM.**

Zie ook: **SELECT_DIAGRAM_TYPE, SELECT_DIAGRAM, NEXT_DIAGRAM, PRINT_DIAGRAM_NAME, PRINT_DIAGRAM_TYPE, OUTPUT,**

PRINT_ADDITIONAL_DIAGRAM.

PRINT_DIAGRAM_NAME

Syntax: PRINT_DIAGRAM_NAME

Functie: afdrukken van de naam van het geselecteerde diagram. (SDW)

Werking: Met het commando **PRINT_DIAGRAM_NAME** wordt de naam van het geselecteerde diagram geschreven naar het geselecteerde uitvoerkanaal of naar een variabele. In beide gevallen dient het diagram eerst geselecteerd te zijn (met **SELECT_DIAGRAM** of **NEXT_DIAGRAM**). Is dat nog niet gebeurd, dan volgt een foutmelding.
De werking van het commando is als volgt voor te stellen:
Als een diagram wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_DIAGRAM_NAME**-commando wordt gegeven, wordt gekeken bij welke component deze markering staat, en wordt vervolgens de naam van dit diagram geschreven. Door een ander diagram te selecteren, verschuift de markering, waardoor het **PRINT_DIAGRAM_NAME**-commando een andere diagramnaam zal schrijven.

Via het **SET**-commando kan een diagramnaam ook als waarde aan een variabele worden toegekend. Het script:

```
select_diagram_type Data Flow Diagram
set $aantal 0
while next_diagram
    BEGIN
    add $aantal 1
    set $dia print_diagram_name
    print $aantal - $dia
    END
```

zal alle namen van Data Flow Diagrams naar het current uitvoerkanaal schrijven, met een nummer ervoor, bijvoorbeeld:

```
1 - uitvoeren
2 - invoeren
```

Aangezien de naam van het actieve diagram door SDWrite

automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWdiagram kan in plaats van het commando **PRINT_DIAGRAM_NAME** ook altijd gebruik gemaakt worden van:

```
print $SDWdiagram
```

Zie ook: **SELECT_DIAGRAM, PRINT_DIAGRAM_TYPE, NEXT_DIAGRAM, SELECT_DIAGRAM_TYPE, PRINT_DIAGRAM.**

PRINT_DIAGRAM_TYPE

Syntax: PRINT_DIAGRAM_TYPE

Functie: afdrukken van het diagram-type van het geselecteerde diagram. (SDW)

Werking: Met het commando **PRINT_DIAGRAM_TYPE** wordt het geselecteerde diagram-type geschreven naar het geselecteerde uitvoerkanaal of naar een variabele. In beide gevallen dient het diagram-type eerst geselecteerd te zijn (dit kan met **SELECT_DIAGRAM_TYPE** of **NEXT_DIAGRAM_TYPE**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een diagram-type wordt geselecteerd, wordt dit door SDWrite als het ware van een markering voorzien. Indien het **PRINT_DIAGRAM_TYPE**-commando wordt gegeven, wordt gekeken bij welk diagram-type deze markering staat, en wordt vervolgens de naam van dit diagram-type geschreven. Door een ander diagram-type te selecteren, verschuift de markering, waardoor het commando **PRINT_DIAGRAM_TYPE** een ander diagram-type zal schrijven.

In combinatie met het **SET**-commando kan de naam van een diagram-type ook als waarde aan een variabele worden toegekend.

Het script:

```
sdwsystem data
set $aantal 0
while next_diagram_type
    BEGIN
    add $aantal 1
    set $type print_diagram_type
    print $aantal - $type
    END
```

zal alle diagram-typen van het SDW-systeem 'data' naar het actieve uitvoerkanaal schrijven, met een nummer ervoor. De

uitvoer kan er als volgt uitzien:

```
1 - Data Flow Diagram
2 - Entity Relationship Diagram
3 - State Transition Diagram
4 - Structure Chart
```

Aangezien de naam van het actieve diagram-type door SDWrite automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWdiagram_type kan in plaats van het commando **PRINT_DIAGRAM_TYPE** ook altijd gebruik gemaakt worden van:

```
print $SDWdiagram_type
```

Zie ook: **NEXT_DIAGRAM_TYPE, SELECT_DIAGRAM_TYPE, SET.**

PRINT_GENERATED_ITEM_HEADER

Syntax: PRINT_GENERATED_ITEM_HEADER

Functie: schrijft de header van de geselecteerde gegenereerde rubriek naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het **PRINT_GENERATED_ITEM_HEADER**-commando wordt de header van de geselecteerde rubriek geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de rubriek eerst geselecteerd te zijn (dit kan zowel met **SELECT_GENERATED_ITEM** als met het commando **NEXT_GENERATED_ITEM**).
Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubriek wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_GENERATED_ITEM_HEADER** wordt gegeven, wordt gekeken bij welke gegenereerde rubriek deze markering staat, en wordt vervolgens de header van deze rubriek geschreven.
Door een andere gegenereerde rubriek te selecteren, verschuift de markering, waardoor een andere rubrieksheader geschreven zal worden met **PRINT_GENERATED_ITEM_HEADER**.

In combinatie met het **SET**-commando kan een rubrieksheader ook als waarde aan een variabele worden toegekend.

De header van een (gegenereerde) rubriek moet niet verward worden met de **HEADER** van een SDWrite-rapport, zoals beschreven bij het keyword **HEADER**.

Zie ook: **NEXT_GENERATED_ITEM**, **PRINT_GENERATED_ITEM_NAME**, **PRINT_GENERATED_ITEM_TEXT**, **SELECT_GENERATED_ITEM**, **SET**.

PRINT_GENERATED_ITEM_NAME

Syntax: PRINT_GENERATED_ITEM_NAME

Functie: schrijft de naam van de geselecteerde gegenereerde rubriek naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het **PRINT_GENERATED_ITEM_NAME**-commando wordt de naam van de geselecteerde rubriek geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de rubriek eerst met **SELECT_GENERATED_ITEM** of **NEXT_GENERATED_ITEM** geselecteerd te zijn. Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubriek wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_GENERATED_ITEM_NAME** wordt gegeven, dan wordt gekeken bij welke gegenereerde rubriek deze markering staat, en wordt vervolgens de naam van deze rubriek geschreven. Door een andere gegenereerde rubriek te selecteren, verschuift de markering, waardoor het **PRINT_GENERATED_ITEM_NAME**-commando een andere rubrieksnaam zal schrijven.

In combinatie met het **SET**-commando kan de naam van een rubriek ook als waarde aan een variabele worden toegekend.

Aangezien de naam van de actieve gegenereerde rubriek door SDWrite automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWgenerated_item kan in plaats van het commando **PRINT_GENERATED_ITEM_NAME** ook altijd gebruik gemaakt worden van:

```
print $SDWgenerated_item
```

Zie ook: **NEXT_GENERATED_ITEM, PRINT_GENERATED_ITEM_HEADER, PRINT_GENERATED_ITEM_TEXT, SELECT_GENERATED_ITEM, SET.**

PRINT_GENERATED_ITEM_TEXT

Syntax: PRINT_GENERATED_ITEM_TEXT [naam | $naam]

Functie: schrijft de geselecteerde tekstregel van de geselecteerde gegenereerde rubriek naar het current uitvoerkanaal of naar een variabele.
of
schrijft de gehele inhoud van de gegenereerde rubriek naar het current uitvoerkanaal. (SDW)

Werking: Om de inhoud van een gegenereerde rubriek af te drukken kan gekozen worden voor:

- Het met één commando afdrukken van de gehele gegenereerde rubriek, door achter het keyword **PRINT_GENERATED_ITEM_TEXT** de naam van de gewenste gegenereerde rubriek op te nemen (eventueel als variabele).

- Het regel voor regel afdrukken van de gegenereerde rubriek door **PRINT_GENERATED_ITEM_TEXT** (zonder argument) in een **WHILE**-constructie op te nemen.

Afdrukken van de gehele rubriek in één keer

Het afdrukken van een de gehele inhoud van een gegenereerde rubriek 'A' kan worden uitgevoerd door:

```
set $gen_naam print A
print_generated_item_text $gen_naam
```

of

```
print_generated_item_text A
```

Het is in dit geval niet nodig de rubriek voorafgaand aan het afdrukken eerst te selecteren. De combinatie van het keyword **PRINT_GENERATED_ITEM_TEXT** en de rubrieksnaam verzorgt namelijk automatisch de selectie van de rubriek.

Het is in principe ook mogelijk de gehele inhoud van de rubriek naar een variabele te schrijven door het commando te combineren met een **SET**-constructie.
Bijvoorbeeld:

```
set $inhoud print_generated_item_text A
```

Deze constructie is echter uitsluitend zinvol bij rubrieken die maximaal **één regel** kunnen bevatten.

Regel voor regel afdrukken van de rubriek

Het **PRINT_GENERATED_ITEM_TEXT**-commando zònder argument zorgt ervoor dat de actieve regel van de gegenereerde rubriek wordt afgedrukt.
Deze regel dient in dit geval **wel** voorafgaand geselecteerd te zijn (met het commando **NEXT_GENERATED_ITEM_TEXT**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubrieksregel wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_GENERATED_ITEM_TEXT** wordt gegeven, dan wordt gekeken bij welke rubrieksregel deze markering staat, en wordt deze regel vervolgens geschreven. Door de volgende rubrieksregel te selecteren, verschuift de markering, waardoor het commando **PRINT_GENERATED_ITEM_TEXT** de volgende rubrieksregel zal schrijven.

Om een hele rubrieksinhoud te schrijven, dient in dit geval gebruik gemaakt te worden van een **WHILE**-constructie:

```
while next_generated_item_text
    print_generated_item_text
```

Door deze constructie wordt elke regel van de geselecteerde gegenereerde rubriek geschreven.

In combinatie met het **SET**-commando kan een rubrieksregel ook als waarde aan een variabele worden toegekend. In het volgende voorbeeld wordt elke regel eerst in de variabele

$regel opgenomen, waarbij uitsluitend regels worden afgedrukt die meer of minder dan 32 tekens bevatten:

```
while next_generated_item_text
    BEGIN
    set $regel print_generated_item_text
    set $len length $regel
    if NOT equal $len 32
        print $regel
    END
```

Zie ook: **SET, NEXT_GENERATED_ITEM, NEXT_GENERATED_ITEM_TEXT, PRINT_GENERATED_ITEM_NAME, SELECT_GENERATED_ITEM PRINT_GENERATED_ITEM_HEADER, PRINT_COMPOSITE.**

PRINT_HEADER

Syntax: PRINT_HEADER

Functie: forceert het schrijven van de **HEADER** naar het current uitvoerkanaal. (Lay Out)

Werking: De SDWrite-header (gedefinieerd met het **HEADER**-commando) wordt automatisch uitgevoerd indien de opgegeven pagina-lengte (opgegeven met **PAGELENGTH**) wordt overschreden.
In sommige gevallen kan het echter wenselijk zijn dat deze **HEADER** ook op andere momenten wordt uitgevoerd. Dit kan op twee manieren: voorwaardelijk met het **NEED**-commando, of onvoorwaardelijk met **PRINT_HEADER**.

Een voor de hand liggende plaats voor **PRINT_HEADER** is voorafgaand aan de eerste schrijf-opdrachten. Zodoende wordt namelijk ook de eerste pagina van een rapport of overzicht van de header voorzien, hetgeen anders niet zou gebeuren (de pagina-lengte is dan immers niet overschreden).

PRINT_HEADER moet niet verward worden met de opdrachten waarmee de header van een SDW-rubriek kan worden geprint: **PRINT_ITEM_HEADER** en **PRINT_GENERATED_ITEM_HEADER**.

Zie ook: **PRINT_GENERATED_ITEM_HEADER**, **PRINT_ITEM_HEADER**, **HEADER**, **PAGELENGTH**.

PRINT_ITEM_DETAILS

Syntax: PRINT_ITEM_DETAILS

Functie: schrijft informatie over het type van de geselecteerde rubriek naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het **PRINT_ITEM_DETAILS**-commando wordt informatie over het type van de geselecteerde rubriek geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de rubriek eerst geselecteerd te zijn (met **SELECT_ITEM** of met **NEXT_ITEM**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubriek wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_ITEM_DETAILS** wordt gegeven, wordt gekeken bij welke vrij definieerbare rubriek deze markering staat, en wordt vervolgens de type-informatie van deze rubriek geschreven. Door een andere vrij definieerbare rubriek te selecteren, verschuift de markering, waardoor het **PRINT_ITEM_DETAILS**-commando de type-informatie van een andere rubriek zal schrijven.

De type-informatie bestaat kan uit de volgende onderdelen bestaan:

- rubriekstype
- optioneel of niet optioneel
- nadere type-aanduiding
- lengte van een vaste-lengte-rubriek

Welke informatie bij een bepaalde rubriek getoond zijn, is afhankelijk van het rubriekstype.

Kolom 1: het rubriekstype

De eerste letter geeft het type aan zoals dat via 'Onderhoud

SE' is gedefinieerd:

a associatie
c samenstelling
e externe file
f vaste lengte
m multiple choice meervoudig
n multiple choice enkelvoudig
o objectklasse
s script
t vrije tekst

Kolom 2: optioneel of niet-optioneel

In de tweede kolom wordt aangegeven of een rubriek via 'Onderhoud SE' als optioneel is gedefinieerd:

n niet-optioneel
o optioneel

Kolom 3: nadere type-aanduiding

In de derde kolom staat bij associatie-rubrieken, vaste-lengte-rubrieken, externe-file-rubrieken en samenstellingsrubrieken van welk type deze zijn:

Associatie-rubrieken:

0 1:1
1 1:N
2 N:1
3 N:M

Vaste-lengte-rubrieken:

a alfanumeriek
n numeriek

Externe-file-rubriek:

o readOnly
w readWrite

Samenstellings-rubriek:

s SDW
m deMarco

Kolom 4: lengte van vaste-lengte-rubriek

De vierde kolom geeft de lengte aan zoals die met 'Onderhoud SE' voor een vaste-lengte-rubriek is ingesteld.

In combinatie met het **SET**-commando kan de type-informatie ook als waarde aan een variabele worden toegekend.

Zie ook: **PRINT_ITEM_HEADER, PRINT_ITEM_NAME, SET.**

PRINT_ITEM_HEADER

Syntax: PRINT_ITEM_HEADER

Functie: schrijft de header van de geselecteerde rubriek naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het **PRINT_ITEM_HEADER**-commando wordt de header van de geselecteerde rubriek geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de rubriek eerst geselecteerd te zijn (met **SELECT_ITEM** of met **NEXT_ITEM**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubriek wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_ITEM_HEADER** wordt gegeven, wordt gekeken bij welke vrij definieerbare rubriek deze markering staat, en wordt vervolgens de header van deze rubriek geschreven. Door een andere vrij definieerbare rubriek te selecteren, verschuift de markering, waardoor het **PRINT_ITEM_HEADER**-commando een andere rubrieksheader zal schrijven.

In combinatie met het **SET**-commando kan een rubrieksheader ook als waarde aan een variabele worden toegekend.

De header van een (gegenereerde) rubriek moet niet verward worden met de **HEADER** van een SDWrite-rapport.

Zie ook: **PRINT_GENERATED_ITEM_HEADER**, **PRINT_HEADER**, **SET**.

PRINT_ITEM_NAME

Syntax: PRINT_ITEM_NAME

Functie: schrijft de naam van de geselecteerde rubriek naar het current uitvoerkanaal of naar een variabele. (SDW)

Werking: Met het commando **PRINT_ITEM_NAME** wordt de naam van de geselecteerde rubriek geschreven naar het current uitvoerkanaal of naar een variabele. In beide gevallen dient de rubriek eerst geselecteerd te zijn (met **SELECT_ITEM** of met **NEXT_ITEM**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubriek wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_ITEM_NAME** wordt gegeven, dan wordt gekeken bij welke vrij definieerbare rubriek deze markering staat, en wordt vervolgens de naam van deze rubriek geschreven. Door een andere vrij definieerbare rubriek te selecteren, verschuift de markering, waardoor het **PRINT_ITEM_NAME**-commando een andere rubrieksnaam zal schrijven.

In combinatie met het **SET**-commando kan de naam van een rubriek ook als waarde aan een variabele worden toegekend.

Aangezien de naam van de actieve rubriek door SDWrite automatisch wordt opgeslagen in de standaard SDWrite-variabele $SDWitem kan in plaats van het commando **PRINT_ITEM_NAME** ook altijd gebruik gemaakt worden van:

```
print $SDWitem
```

Zie ook: **PRINT_ITEM_HEADER**, **SET**, **PRINT_GENERATED_ITEM_NAME**, **PRINT_ITEM_TEXT**.

PRINT_ITEM_TEXT

Syntax: PRINT_ITEM_TEXT [rubrieksnaam | $rubrieksnaam]

Functie: schrijft de geselecteerde regel van de geselecteerde rubriek naar het current uitvoerkanaal of een variabele.
of
schrijft de gehele rubrieksinhoud naar het actieve uitvoerkanaal. (SDW)

Werking: Om de inhoud van een rubriek af te drukken kan gekozen worden voor:

- Het met één commando afdrukken van de gehele rubriek, door achter het keyword **PRINT_ITEM_TEXT** de naam van de gewenste rubriek op te nemen (eventueel als variabele)

- Het regel voor regel afdrukken van de rubriek door **PRINT_ITEM_TEXT** (zonder argument) in een **WHILE**-constructie op te nemen.

Afdrukken van de gehele rubriek in één keer

Het afdrukken van een de gehele inhoud van een rubriek 'A' kan worden uitgevoerd door:

```
set $rub_naam print A
print_item_text $rub_naam
```

of

```
print_item_text A
```

Het is in dit geval niet nodig de rubriek voorafgaand aan het afdrukken eerst te selecteren. De combinatie van het keyword **PRINT_ITEM_TEXT** en de rubrieksnaam verzorgt namelijk automatisch de selectie van de rubriek.

Het is in principe ook mogelijk de gehele inhoud van de

rubriek naar een variabele te schrijven door het commando te combineren met een **SET**-constructie.
Bijvoorbeeld:

```
set $inhoud print_item_text A
```

Deze constructie is echter uitsluitend zinvol bij rubriekstypen die maximaal **één regel** kunnen bevatten, zoals rubrieken van de type Objectklasse, Multiple Choice Enkelvoudig en Vaste lengte.

Regel voor regel afdrukken van de rubriek

Het **PRINT_ITEM_TEXT**-commando zònder argument zorgt ervoor dat de actieve regel van de rubriek wordt afgedrukt.
Deze regel dient in dit geval **wel** voorafgaand geselecteerd te zijn (met het commando **NEXT_ITEM_TEXT**). Is dat nog niet gebeurd, dan volgt een foutmelding.

De werking van het commando is als volgt voor te stellen:
Als een rubrieksregel wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het commando **PRINT_ITEM_TEXT** wordt gegeven, dan wordt gekeken bij welke rubrieksregel deze markering staat, en wordt deze regel vervolgens geschreven. Door de volgende rubrieksregel te selecteren, verschuift de markering, waardoor het commando **PRINT_ITEM_TEXT** de volgende rubrieksregel zal schrijven.

Om een hele rubrieksinhoud te schrijven, dient in dit geval gebruik gemaakt te worden van een **WHILE**-constructie:

```
while next_item_text
    print_item_text
```

Door deze constructie wordt elke regel van de geselecteerde rubriek geschreven.

In combinatie met het **SET**-commando kan een rubrieksregel ook als waarde aan een variabele worden toegekend. In het volgende voorbeeld wordt elke regel eerst in de variabele

$regel opgenomen, waarbij lege regels niet worden afgedrukt:

```
while next_item_text
    BEGIN
    set $regel print_item_text
    set $len length $regel
    if NOT equal $len 32
        print $regel
    END
```

Zie ook: **NEXT_ITEM_TEXT**, **PRINT_GENERATED_ITEM_TEXT**, **SET**, **PRINT_COMPOSITE**.

PRINT_NUMBER

Syntax: PRINT_NUMBER

Functie: schrijft het hiërarchisch volgnummer van de geselecteerde component naar het current uitvoerkanaal of een variabele. (SDW)

Werking: Dit commando kan alleen worden gebruikt voor componenten met een hiërarchisch nummer, zoals Functies of AO-processen. Het commando schrijft het hiërarchische nummer van een component naar het current uitvoerkanaal of een variabele. De component dient hiervoor geselecteerd te zijn. Is dat niet het geval, dan volgt een foutmelding.
Indien de geselecteerde component van een component-type zonder hiërarchische nummering is, dan zal het commando geen resultaat hebben (ook geen foutmelding).

De werking van het commando is als volgt voor te stellen:
Als een component wordt geselecteerd, wordt deze door SDWrite als het ware van een markering voorzien. Indien het **PRINT_NUMBER**-commando wordt gegeven, wordt gekeken bij welke component deze markering staat, en wordt vervolgens het hiërarchische nummer van deze component geschreven. Door een andere component te selecteren, verschuift de markering, waardoor het **PRINT_NUMBER**-commando het nummer van een andere component zal schrijven.

In combinatie met het **SET**-commando kan het nummer van een component ook als waarde aan een variabele worden toegekend.

Het script:

```
select_component_type Functie
while next_component
    BEGIN
    set $comp print_component
    set $num print_number
    newline off
    print $num
```

```
column 16
newline on
print $comp
END
```

zal alle functie-namen (op alfabetische volgorde) naar het current uitvoerkanaal schrijven met het hiërarchische nummer ervoor.
De uitvoer kan er als volgt uitzien:

```
1.2     aanname
1.1     afgifte
1       beheer
1.1.1   distributie
1.1.2   verzending
```

Door deze uitvoer naar een file te schrijven, en deze file vervolgens te gebruiken als invoer in een sorteer-script (**SORT** HIERARCHIE) kan een overzicht van functies gemaakt worden met de functies in hiërarchische volgorde.

Zie ook: **SORT**.

PRINT_PART

Syntax: PRINT_PART <$txt> <$start|start [$end|end]> [/trim]

Functie: print een bepaald gedeelte van een variabele, namelijk vanaf de gegeven startpositie tot en met de gegeven eindpositie, waarbij trailing spaces weggelaten kunnen worden (/trim). (Lay Out)

Werking: SDWrite biedt de gebruiker zowel bij invoer als bij uitvoer de mogelijkheid om slechts een gedeelte van de uitgangstekst in te lezen c.q. weg te schrijven.
Om een gedeelte van een regel in te lezen dient gebruik gemaakt te worden van het commando **LINE**; voor gedeeltelijke uitvoer dient **PRINT_PART**. Met beide commando's kan ook een waarde aan een variabele worden toegekend.

Het **PRINT_PART**-commando heeft twee mogelijke resultaten: er wordt een gedeelte van de opgegeven variabele naar het current uitvoerkanaal of een variabele geschreven òf er wordt niets (een lege string) geschreven.

Een lege string wordt geschreven in de volgende gevallen:

- als de opgegeven startpositie groter is dan de opgegeven eindpositie
- als de opgegeven startpositie groter is dan de lengte van de opgegeven variabele.

In alle andere gevallen zal het aangegeven deel van de opgegeven variabele worden geschreven, vanaf de startpositie tot en met de eindpositie. Indien de lengte van de opgegeven variabele kleiner is dan de opgegeven eindpositie, dan wordt de lengte van de variabele als eindpositie beschouwd.
Indien het gedeelte dat geschreven wordt eindigt op een of meer spaties (zgn. trailing spaces), dan kunnen deze verwijderd worden door /trim aan de opdracht toe te voegen.

Voorbeeld:

Uitgangspunt is de variabele $tekst, die via een gecombineerde **SET / PRINT**-opdracht de waarde 'SDWrite voorbeeld' krijgt (met twee spaties in het midden). Daarachter wordt voor verschillende **PRINT_PART**-opdrachten het resultaat aangegeven. Met <nul> wordt de lege string bedoeld, met [ ] een spatie.

```
set $deel1 SDWrite
set $deel2 voorbeeld
set $tekst print $deel1  $deel2

(* commando *)              (* uitvoer *)
print_part $tekst 1 3        :SDW
print_part $tekst 5 9        :ite[ ][ ]_
print_part $tekst 5 9 /trim  :ite
print_part $tekst 3 1        :<nul>
print_part $tekst 8 9        :[ ][ ]
print_part $tekst 8 9 /trim  :<nul>
print_part $tekst 6 11       :te[ ][]vo
print_part $tekst 17 20      :ld
print_part $tekst 19 21      :<nul>
```

Combinatie van het **PRINT_PART**-commando met het **SET**-commando leidt ertoe dat het resultaat van het **PRINT_PART**-commando als waarde wordt toegekend aan de variabele.

Ook spaties kunnen op deze wijze in een variabele worden opgenomen. Dit is niet mogelijk met een directe toekenning. Directe toekenning van een waarde met een spatie, zoals:

```
set $nieuw SDWrite  voorbeeld
```

leidt er namelijk toe dat alleen de tekst vóór de spatie als waarde aan de variabele wordt toegekend. In dit geval zal $nieuw dus de waarde 'SDWrite' krijgen.

Wordt echter gebruik gemaakt van de opdracht:

```
set $nieuw print_part $tekst 1 18
```

dan zal wèl de waarde 'SDWrite voorbeeld' worden toegekend aan $nieuw.

Zie ook: **LINE, PRINT, SET.**

PROMPT

Syntax: PROMPT [prompt-regel]

Functie: leest een regel van het standaard invoerkanaal en schrijft deze naar het current uitvoerkanaal of een variabele. (IO)

Werking: Om een script interactief te maken, zijn vooral de keywords **DISPLAY** en **PROMPT** van belang.
Als de SDWrite-interpreter het keyword **PROMPT** tegenkomt, wordt de verwerking van een script onderbroken totdat <Enter> wordt ingetoetst.
Hiermee kan een script tijdelijk worden gestopt, bijvoorbeeld om de gebruiker de tijd te geven de informatie op het scherm te lezen.
Voorbeeld:

```
display Druk op <Enter> voor vervolg ...
prompt
```

Het is ook mogelijk het keyword **PROMPT** te gebruiken om tijdens de verwerking van een script invoer door de gebruiker mogelijk te maken.
In het volgende voorbeeld wordt op deze wijze gevraagd welk SDW-systeem het script moet gebruiken:

```
newline off
display Welk systeem ?
spaces 1
newline on
set $sys prompt

sdwsystem $sys
```

Ook in dit geval wordt de verwerking van het script pas vervolgd als <Enter> is ingetoetst. De tekens die daarvoor zijn ingetoetst (de naam van een SDW-systeem), worden als waarde aan de variabele toegekend, waarna die variabele in combinatie met **SDWSYSTEM** gebruikt kan worden om een systeem te selecteren.

Hetzelfde resultaat kan ook bereikt worden door de vraag ('Welk systeem?') als argument achter **PROMPT** te plaatsen. Hierbij wordt de vraag altijd getoond alsof **NEWLINE** 'OFF' staat.
Het script-gedeelte ziet er dan als volgt uit:

```
set $sys prompt Welk systeem ?
sdwsystem $sys
```

Zie ook: **DISPLAY, SET.**

QUIT

Syntax: QUIT

Functie: beëindigt SDWrite. (-)

Werking: Het keyword **QUIT** dient om te stoppen met SDWrite. Het commando is alleen nodig indien online gewerkt wordt. Het is overbodig indien een SDWrite-script wordt gebruikt.

Tegelijk intoetsen van de toetsen <Ctrl> en <z> levert hetzelfde resultaat.

Zie ook: -

SCRIPT

Syntax: SCRIPT <scriptnaam | $scriptnaam>

Functie: aanroepen van een script vanuit een ander script (PRG)

Werking: Met het **SCRIPT**-commando kan vanuit het gestarte script een ander script worden aangeroepen. Daarbij dient rekening gehouden te worden met het directory-pad van het aangeroepen script.
Indien bijvoorbeeld het script 'script2.scr' is opgenomen in de directory 'C:\SDW\SCRIPTS' en dit script moet worden aangeroepen vanuit een ander script (bijvoorbeeld script1.scr), dan dient daartoe in script1.scr het volgende commando te worden opgenomen:

```
script c:\sdw\scripts\script2.scr
```

Een aangeroepen script wordt uitgevoerd **alsof** dat script op de betreffende plaats is opgenomen in het eerste script. De werking van **SCRIPT** is derhalve goed te vergelijken met dat van **MACRO**.
Belangrijk verschil is, dat er bij **MACRO** een essentieel verschil bestaat tussen de macro-definitie en de macro-uitvoering. Bij **SCRIPT** is hiervan geen sprake.

Dankzij het keyword **SCRIPT** kunnen veelgebruikte script-onderdelen in aparte files bewaard worden en in elk gewenst script worden opgenomen zónder dat ze steeds opnieuw dienen te worden ingetoetst.

Zie ook: **MACRO**.

SDWSYSTEM

Syntax: SDWSYSTEM [$variabele | SDW-systeemnaam]

Functie: 1. eerste voorkomen: initialiseert het opgegeven SDW-systeem.
2. alle volgende voorkomens: schrijft de naam van het geselecteerde systeem naar het current uitvoerkanaal of een variabele. (SDW)

Werking: Om gegevens in de Systeem Encyclopedie te benaderen, dient allereerst het SDW-systeem geselecteerd te worden.
Dit commando kan op drie manieren gebruikt worden:

- zonder argument
- met een naam als argument
- met een variabele als argument

Zonder argument

Door het commando:

```
sdwsystem
```

(zonder parameter) wordt het systeem geselecteerd dat het laatst in SDW is geselecteerd, de actieve systeem-directory.

Met een naam als argument

Door achter het keyword direct de gewenste naam op te geven, bijvoorbeeld:

```
sdwsystem data
```

wordt het SDW-systeem met de betreffende naam (in dit geval 'data') geselecteerd. Indien dit systeem niet bekend is, volgt een foutmelding.

Met een variabele als argument

Door gebruik te maken van een variabele als argument, bijvoorbeeld:

```
sdwsystem $data
```

wordt het SDW-systeem geselecteerd met de waarde van de variabele $data als naam. Heeft de variabele (nog) geen waarde of is het geselecteerde systeem onbekend, dan volgt een foutmelding.
Het gebruik van een variabele als argument is vooral handig in interactieve scripts, die daardoor op verschillende SDW-systemen toepasbaar zijn.
De systeem-selectie kan bijvoorbeeld als volgt worden uitgevoerd:

```
display Welk systeem ?
set $sys prompt

sdwsystem $sys
```

In een script kan slechts eenmaal een systeem geselecteerd worden. Elke volgende aanroep van **SDWSYSTEM** leidt ertoe dat de naam van het geselecteerde systeem naar het current uitvoerkanaal wordt geschreven.
Het script:

```
sdwsystem test
sdwsystem
sdwsystem data
```

heeft derhalve als uitvoer:

```
<* selectie van het systeem test *>
test (* schijven van de systeemnaam *)
test (* idem, argument 'data' wordt genegeerd *)
```

Aangezien tijdens het verwerken van een SDW-systeem automatisch de standaard SDWrite-variabele $SDWsystem wordt gevuld, kan de naam van het actieve systeem ook als volgt afgedrukt worden:

```
print $SDWsystem
```

Het commando **SDWSYSTEM** is uitsluitend zinvol binnen de commandline-versie van SDWrite. In alle overige gevallen is het SDW-systeem actief dat met het hulpprogramma 'Selecteer systeem' is geselecteerd.

Zie ook: **DISPLAY**, **PROMPT**.

SELECT_COMPONENT

Syntax: SELECT_COMPONENT <componentnaam | $var>

Functie: selecteert de component met de opgegeven naam, uitgaande van het geselecteerde component-type. (SDW)

Werking: Om gegevens over een component te kunnen benaderen dient deze component eerst geselecteerd te worden. Dit kan op twee manieren: met **NEXT_COMPONENT** (dat bij elke aanroep de volgende component selecteert) of met **SELECT_COMPONENT**.
In beide gevallen **moet** vooraf het component-type geselecteerd zijn.

Het selecteren van componenten in SDWrite is als volgt voor te stellen:
Indien in SDWrite een component wordt geselecteerd, dan wordt bij de betreffende componentnaam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_COMPONENT** op welke component dat commando betrekking heeft.
Het commando **SELECT_COMPONENT** zorgt ervoor dat de markering bij de opgegeven naam wordt geplaatst.
Indien deze componentnaam niet bekend is, volgt een foutmelding.

Het is mogelijk aan **SELECT_COMPONENT** een variabele als parameter mee te geven. Deze variabele dient vooraf met **SET** een bepaalde waarde gekregen te hebben.
Deze mogelijkheid is vooral handig in een interactief script, dat daardoor op verschillende componenten toepasbaar is.

Voorbeeld:

```
display Welke component?
set $comp prompt

select_component $comp
```

Door het commando **SELECT_COMPONENT** wordt automatisch

de standaard SDWrite-variabele $SDWcomponent gevuld. Deze variabele bevat op elk moment de naam van de actieve component.

Zie ook: **NEXT_COMPONENT, PRINT_COMPONENT.**

SELECT_COMPONENT_TYPE

Syntax: SELECT_COMPONENT_TYPE <type | $var>

Functie: selecteert het opgegeven component-type. (SDW)

Werking: Om gegevens over een component-type te kunnen benaderen dient dit component-type eerst geselecteerd te worden.
Dit kan op twee manieren:

- met het commando **NEXT_COMPONENT_TYPE**, dat bij elke aanroep het volgende component-type selecteert.

- met het commando **SELECT_COMPONENT_TYPE**.

Het selecteren van component-typen in SDWrite is als volgt voor te stellen:
Indien in SDWrite een component-type wordt geselecteerd, dan wordt bij de betreffende naam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_COMPONENT_TYPE** op welk component-type dat commando betrekking heeft.
Het commando **SELECT_COMPONENT_TYPE** zorgt ervoor dat de markering bij de opgegeven naam wordt geplaatst.
Indien dit component-type niet bekend is, volgt een foutmelding.

Het is mogelijk aan **SELECT_COMPONENT_TYPE** een variabele als parameter mee te geven. Deze variabele dient vooraf met **SET** een bepaalde waarde gekregen te hebben.
Deze mogelijkheid is vooral handig in een interactief script, dat daardoor op verschillende component-typen toepasbaar is.

Voorbeeld:

```
display Welk component-type?
set $type prompt

select_component_type $type
```

★ Door **SELECT_COMPONENT_TYPE** wordt automatisch de standaard SDWrite-variabele $SDWcomponent_type gevuld. Deze variabele bevat op elk moment de naam van het actieve component-type.

Zie ook: **NEXT_COMPONENT_TYPE, PRINT_COMPONENT_TYPE.**

SELECT_DIAGRAM

Syntax: SELECT_DIAGRAM <naam | $naam>

Functie: selecteert het diagram met de opgegeven naam, uitgaande van het geselecteerde diagram-type. (SDW)

Werking: Om gegevens over een diagram te kunnen benaderen dient dit diagram eerst geselecteerd te worden. Dit kan op twee manieren: met **NEXT_DIAGRAM** (dat bij elke aanroep het volgende diagram selecteert) of met **SELECT_DIAGRAM**.
In beide gevallen **moet** vooraf het diagram-type geselecteerd zijn.

Het selecteren van diagrammen in SDWrite is als volgt voor te stellen:
Indien in SDWrite een diagram wordt geselecteerd, dan wordt bij de betreffende diagramnaam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_DIAGRAM** op welk diagram dat commando betrekking heeft.
Het commando **SELECT_DIAGRAM** zorgt ervoor dat de markering bij de opgegeven naam wordt geplaatst. Indien deze diagramnaam niet bekend is, volgt een foutmelding.

Het is mogelijk aan **SELECT_DIAGRAM** een variabele als parameter mee te geven. Deze variabele dient vooraf met **SET** een bepaalde waarde gekregen te hebben.
Deze mogelijkheid is vooral handig in een interactief script, dat daardoor op verschillende diagrammen toepasbaar is.
Voorbeeld:

```
display Welk diagram?
set $dia prompt
select_diagram $dia
```

Door **SELECT_DIAGRAM** wordt automatisch de standaard SDWrite-variabele $SDWdiagram gevuld. Deze variabele bevat op elk moment de naam van het actieve diagram.

Zie ook: **SELECT_DIAGRAM_TYPE, PRINT_DIAGRAM, NEXT_DIAGRAM, PRINT_DIAGRAM_NAME.**

SELECT_DIAGRAM_TYPE

Syntax: SELECT_DIAGRAM_TYPE <type | $type>

Functie: selecteert het opgegeven diagram-type. (SDW)

Werking: Om gegevens over een diagram-type te kunnen benaderen dient dit diagram-type eerst geselecteerd te worden. Dit kan op twee manieren: met het commando **NEXT_DIAGRAM_TYPE** (dat bij elke aanroep het volgende diagram-type selecteert) of met het commando **SELECT_DIAGRAM_TYPE**.

Het selecteren van diagram-typen in SDWrite is als volgt voor te stellen:
Indien in SDWrite een diagram-type wordt geselecteerd, dan wordt bij de betreffende naam als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_DIAGRAM_TYPE** op welk diagram-type dat commando betrekking heeft.
Het commando **SELECT_DIAGRAM_TYPE** zorgt ervoor dat de markering bij de opgegeven naam wordt geplaatst.
Indien dit diagram-type niet bekend is, volgt een foutmelding.

Het is mogelijk aan **SELECT_DIAGRAM_TYPE** een variabele als parameter mee te geven. Deze variabele dient vooraf met **SET** een bepaalde waarde gekregen te hebben.
Deze mogelijkheid is vooral handig in een interactief script, dat daardoor op verschillende diagram-typen toepasbaar is.

Voorbeeld:

```
display Welk diagram-type?
set $type prompt
select_diagram_type $type
```

Door **SELECT_DIAGRAM_TYPE** wordt automatisch de standaard SDWrite-variabele $SDWdiagram_type gevuld. Deze variabele bevat op elk moment de naam van het actieve diagram-type.

Zie ook: **NEXT_DIAGRAM_TYPE, PRINT_DIAGRAM_TYPE.**

SELECT_GENERATED_ITEM

Syntax: SELECT_GENERATED_ITEM <rubriek>

Functie: selecteert, uitgaande van het geselecteerde component-type, de gegenereerde rubriek met de opgegeven naam. (SDW)

Werking: Om gegevens in een gegenereerde rubriek te kunnen benaderen dient deze rubriek eerst geselecteerd te worden. Dit kan op twee manieren: met **NEXT_GENERATED_ITEM** (dat bij elke aanroep de volgende gegenereerde rubriek selecteert) of met **SELECT_GENERATED_ITEM.**
In beide gevallen **moet** vooraf het component-type geselecteerd zijn. Of vervolgens eerst de component en dan de rubriek, of eerst de rubriek en dan de component geselecteerd wordt, is niet van belang. Om de inhoud van een rubriek te benaderen (met **NEXT_GENERATED_ITEM_TEXT**) dienen echter beide geselecteerd te zijn.

Het selecteren van rubrieken in SDWrite is als volgt voor te stellen:
Indien in SDWrite een rubriek wordt geselecteerd, dan wordt bij de betreffende rubriek als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_GENERATED_ITEM_HEADER** op welke rubriek dat commando betrekking heeft.
Het commando **SELECT_GENERATED_ITEM** zorgt ervoor dat de markering bij de opgegeven naam wordt geplaatst.
Indien deze rubrieksnaam niet bekend is, volgt een foutmelding.

Het is mogelijk aan **SELECT_GENERATED_ITEM** een variabele als parameter mee te geven. Deze variabele dient vooraf met **SET** een bepaalde waarde gekregen te hebben. Dit is vooral handig in een interactief script, dat daardoor op verschillende gegenereerde rubrieken toepasbaar is, bijvoorbeeld:

```
display Welke rubriek?
set $g_item prompt
select_generated_item $g_item
```

Door **SELECT_GENERATED_ITEM** wordt automatisch de standaard SDWrite-variabele $SDWgenerated_item gevuld. Deze variabele bevat op elk moment de naam van de actieve gegenereerde rubriek.

Zie ook: **NEXT_GENERATED_ITEM.**

SELECT_ITEM

Syntax: SELECT_ITEM <rubriek>

Functie: selecteert, uitgaande van het geselecteerde component-type, de rubriek met de opgegeven naam. (SDW)

Werking: Om gegevens in een vrij definieerbare rubriek te kunnen benaderen dient deze rubriek eerst geselecteerd te worden.
Dit kan op twee manieren: met **NEXT_ITEM** (dat bij elke aanroep de volgende vrij definieerbare rubriek selecteert) of met **SELECT_ITEM**.
In beide gevallen **moet** vooraf het component-type geselecteerd zijn. Of vervolgens eerst de component en dan de rubriek, of eerst de rubriek en dan de component geselecteerd wordt, is niet van belang. Om de inhoud van een rubriek te benaderen (met **NEXT_ITEM_TEXT**) dienen echter beide geselecteerd te zijn.

Het selecteren van rubrieken in SDWrite is als volgt voor te stellen:
Indien in SDWrite een rubriek wordt geselecteerd, wordt bij de betreffende rubriek als het ware een markering geplaatst. Dankzij die markering weet SDWrite bij commando's als **PRINT_ITEM_HEADER** op welke rubriek dat commando betrekking heeft.
Het commando **SELECT_ITEM** zorgt ervoor dat de markering bij de opgegeven naam wordt geplaatst.
Indien deze naam niet bekend is, volgt een foutmelding.

Het is mogelijk aan **SELECT_ITEM** een variabele als parameter mee te geven. Deze variabele dient vooraf met **SET** een bepaalde waarde gekregen te hebben. Dit is vooral handig in een interactief script, dat daardoor op verschillende vrij definieerbare rubrieken toepasbaar is.
Bijvoorbeeld:

```
display Welke rubriek?
set $item prompt
select_item $item
```

Door **SELECT_ITEM** wordt automatisch de standaard SDWrite-variabele $SDWitem gevuld. Deze variabele bevat op elk moment de naam van de actieve rubriek.

Zie ook: **NEXT_ITEM**.

SELECT_SDWDRIVER

Syntax: SELECT_SDWDRIVER <driver-naam>

Functie: selecteert de gewenste SDW-printerdriver uit de LIB-directory. (IO)

Werking: Default schrijft SDWrite alle uitvoer naar het beeldscherm. Met het keyword **OUTPUT** kan de uitvoer echter ook gestuurd worden naar andere uitvoerkanalen, zoals een printer of een file. Bovendien kan worden aangegeven of bij de uitvoer gebruik gemaakt dient te worden van een SDW-printerdriver of dat de uitvoer als 'platte' ASCII-file geschreven wordt.

Indien wordt gekozen voor aansturing via een printerdriver, wordt standaard gebruik gemaakt van de printerdriver die is geselecteerd bij het installeren van het werkstation (sdw_drv.mpt in de LIBWS-directory).
Indien bij de installatie van SDW ook andere printerdrivers zijn geïnstalleerd, kan echter ook elk van deze andere drivers worden gekozen. Deze drivers bevinden zich in de LIB-directory en hebben als extensie '.MPT'.

Voorbeeld:
Een script dient de uitvoer te schrijven in de file 'uitvoer'. Deze uitvoer dient in het printer-formaat van de printer 'abc' te worden aangemaakt. In de LIB-directory staat de file 'abc.mpt'.
Het script dient nu in elk geval de volgende regels te bevatten:

```
select_sdwdriver abc
output uitvoer SDWDRIVER
```

★ Vanzelfsprekend dient het selecteren van de driver voorafgaand aan het **OUTPUT**-commando te gebeuren.

★ Indien een **diagram** wordt afgedrukt, wordt **automatisch** gebruik gemaakt van de geselecteerde printerdriver.

Zie ook: **OUTPUT**.

SEMICOLON

Syntax: SEMICOLON <ON | OFF>

Functie: zorgt ervoor dat een ; (semicolon) aan het einde van een **PRINT**-opdracht wordt geïnterpreteerd als een teken in de te **PRINT**-en tekst òf als aanduiding dat geen newline moet worden gegenereerd. **SEMICOLON** werkt als een zogenaamde toggle. (Lay Out)

Werking: Standaard wordt elke SDWrite-schrijfopdracht (alle print-opdrachten, date, time, pageno, etc.) gevolgd door een newline. Uitzondering hierop vormen **PRINT**-opdrachten die eindigen op een puntkomma (; ofwel: semicolon). Deze **PRINT**-opdrachten worden standaard **zonder** de puntkomma en **zonder** newline geschreven, ongeacht of 'newline' met behulp van het **NEWLINE**-commando op ON of OFF staat.

Met **SEMICOLON** 'OFF' kan aangegeven worden dat ook dergelijke **PRINT**-opdrachten als 'normale' opdrachten geïnterpreteerd dienen te worden.

Indien het gewenst is dat een afsluitende puntkomma als onderdeel van de te schrijven tekst wordt beschouwd, dan kan dat kenbaar gemaakt worden door de opdracht:

```
semicolon off
```

Vanaf deze opdracht zullen alle **PRINT**-opdrachten die eindigen op een puntkomma 'normaal' worden uitgevoerd (dus: met een afsluitende puntkomma en, afhankelijk van de instelling van **NEWLINE**, al dan niet met een newline).
Komt in het vervolg van het script de volgende opdracht voor:

```
semicolon on
```

dan zullen vanaf deze opdracht alle **PRINT**-opdrachten weer 'normaal' worden geïnterpreteerd, dat wil zeggen **met** puntkomma en afhankelijk van de instelling van **NEWLINE** met of zonder newline.

Voorbeeld:

```
newline on
print SDWrite1/;
print SDWrite2/;
semicolon on
print SDWrite3/;
newline off
print SDWrite4/;
semicolon off
print SDWrite5/;
newline on
print SDWrite6/;
print SDWrite7/;
```

heeft als uitvoer:

```
SDWrite1/;
SDWrite2/;
SDWrite3/SDWrite4/ SDWrite5/;SDWrite6/;
SDWrite7/;
```

Zie ook: **NEWLINE, PRINT.**

SET

Syntax: SET <$varnaam> <waarde>

Functie: kent de gegeven waarde toe aan de gegeven variabele. (PRG)

Werking: Om met een variabele te kunnen werken, dient deze altijd eerst een waarde te krijgen. Hiervoor dient in SDWrite het **SET**-commando, dat diverse vormen kent.
De basisvorm is als volgt:

```
set $sys test
```

waardoor aan de variabele $sys de waarde 'test' wordt toegekend.
Het is **niet** mogelijk op deze wijze een variabele met een of meer spaties te initialiseren. De opdracht:

```
set $regel 001; Functie
```

kent aan de variabele $regel alleen de tekens tot aan de eerste spatie toe, dus: '001;'. Alles na de eerste spatie wordt genegeerd.

SET kan echter ook gecombineerd worden met SDWrite-schrijfopdrachten zoals **PRINT**, **DATE** en **SDWSYSTEM**.
Resultaat is dat aan de variabele de uitvoer van de schrijfopdracht wordt toegekend (respectievelijk de geprinte tekst, de datum of de naam van het geselecteerde SDW-systeem).

In combinatie met een **PRINT**-opdracht kan aan een variabele **wel** een waarde met een of meer spaties worden toegekend.

De opdrachten:

```
set $a SDWrite-
set $b voorbeeld

set $var1 print $a $b
set $var2 print $a$b
```

hebben als resultaat:

```
$var1   = SDWrite- voorbeeld
$var2   = SDWrite-voorbeeld
```

waarbij in $var1 dus wel een spatie voorkomt.

Zie ook: **ADD, DIVIDE, MULTIPLY, PRINT, PRINT_PART, SUBTRACT.**

SORT

Syntax: **a.** SORT < ON | OFF >
b. SORT <start eind>
c. SORT [HIERARCHIE]

Functie: **a.** schrijft alle uitvoer naar een tussenfile in plaats van naar het current uitvoerkanaal (ON), waarna de gegevens in de tussenfile gesorteerd worden weggeschreven naar het current uitvoerkanaal (OFF).
b. sorteert (als **SORT** ON staat) de regels in de tussenfile vanaf positie start tot en met positie eind.
c. sorteert (als **SORT** ON staat) de regels in de tussenfile hiërarchisch. Indien dit commando wordt weggelaten, wordt alfabetisch gesorteerd.

Werking: Het sorteren van gegevens gaat in SDWrite als volgt:
Allereerst dient met **SORT** ON aangegeven te worden dat er gesorteerd dient te worden. Vanaf dat moment wordt alle uitvoer niet langer naar het current uitvoerkanaal geschreven, maar naar een aparte tussenfile ('sortfile').
Zijn alle te sorteren gegevens geschreven, dan dient het commando **SORT** OFF gegeven te worden. Op dat moment wordt de tussenfile gesorteerd en wordt de uitvoer alsnog (gesorteerd) naar het current uitvoerkanaal geschreven.

Standaard sorteert SDWrite alfabetisch en vanaf de eerste positie van een regel.

Een voorbeeld: Stel dat in een file ('invoer.dat') de volgende gegevens staan:

```
dfd -  Function Analysis
sch -  Structured Design
swr -  SDWrite
i-a -  Information Analysis
nor -  Data Analysis
```

en het is de bedoeling de gegevens in deze file in alfabetische volgorde naar het beeldscherm te schrijven, waarbij het voorvoegsel (de tekst voor de '-') wordt genegeerd.

Hiervoor kan het volgende script gebruikt worden:

```
line_open invoer.dat        (* 1 *)
sort on                     (* 2 *)

while set $regel line       (* 3 *)
    print_part $regel 7     (* 4 *)

sort off                    (* 5 *)
```

Met opdracht (1) wordt de invoerfile geopend voor verwerking. Vervolgens wordt aangegeven dat gesorteerd moet worden (2).
Pas daarna volgen de schrijfcommando's waarop het sorteren betrekking heeft: zolang er nog een $regel kan worden ingelezen (3), moet deze regel vanaf het zevende teken worden geschreven (4).
Omdat **SORT** inmiddels ON staat, wordt deze uitvoer echter niet direct naar het current uitvoerkanaal geschreven, maar achtergehouden in een tussenfile.
Pas op het moment dat met **SORT** OFF aangegeven wordt dat er geen regels meer gesorteerd hoeven te worden (5), wordt deze tussenfile gesorteerd en weggeschreven naar het current uitvoerkanaal.
De uitvoer ziet er als volgt uit:

```
Data Analysis
Function Analysis
Information Analysis
SDWrite
Structured Design
```

Merk hierbij op dat sorteren in SDWrite altijd betrekking heeft op de weggeschreven regels (in het voorbeeld dus vanaf positie 7) en **niet** op de ingelezen regels! Het is altijd de **uitvoer** die naar tussenfile wordt geschreven en wordt gesorteerd!

Het is ook mogelijk een andere sorteerwijze te gebruiken dan alfabetisch en vanaf positie 1.

Allereerst kan gesorteerd worden op basis van een begin- en een eindpositie in de geschreven regels.

Stel dat uitgaande van het voorbeeld dezelfde sorteer-volgorde gewenst is, maar het voorvoegsel moet wèl meegeschreven worden.
In dat geval dient elke ingelezen $regel geheel te worden weggeschreven, maar dient deze uitvoer gesorteerd te worden vanaf positie 7. Hierbij dient ook altijd een eindpositie te worden meegegeven (bijvoorbeeld 80):

```
line_open invoer.dat      (* 1 *)
sort on                   (* 2 *)
sort 7 80                 (* X *)

while set $regel line     (* 3 *)
    print $regel          (* 4 *)

sort off                  (* 5 *)
```

In dit geval wordt elke $regel geheel weggeschreven (4), waarbij de gewenste sorteervolgorde wordt gegarandeerd door de extra opdracht (X).
De uitvoer ziet er nu als volgt uit:

```
nor  -  Data Analysis
dfd  -  Function Analysis
i-a  -  Information Analysis
swr  -  SDWrite
sch  -  Structured Design
```

De opdracht **SORT** <begin eind> kan niet gebruikt worden als vervanging van **SORT** ON, maar dient altijd aan dit commando te worden toegevoegd!

Merk bovendien op dat het commando **SORT** x y altijd betrekking heeft op de uitvoer. Gegeven de file 'test.txt':

```
abcdefghij
0123456789
```

zal het script:

```
line_open test.txt
sort on
sort 2 80                 (* a *)
```

```
while set $regel line 3          (* b *)
    print_part $regel 2 5        (* c *)

sort off
```

als volgt werken:
Elke regel wordt ingelezen vanaf positie 3 (b), met als resultaat de regels:

```
cdefghij
23456789
```

Vervolgens worden deze regels (naar tussenfile) geschreven van positie 2 tot en met positie 5. De uitvoer daarvan is:

```
defg
3456
```

Tenslotte worden deze regels gesorteerd vanaf positie 2, dus op basis van:

```
efg
456
```

De uitvoer is dus uiteindelijk:

```
3456
defg
```

De tweede mogelijk sorteervolgorde is de hiërarchische volgorde.

Voorbeeld: Er is een file 'functies.abc', die uitvoer is van een script dat alle Functie-namen schrijft, voorafgegaan door de aanduiding 'Func' + een interne drieletterige code, en op kolom 34 gevolgd door het hiërarchische nummer (zie hiervoor het keyword **PRINT_NUMBER**).

De file 'functies.abc' zie er als volgt uit:

```
FuncAdm: aanname  1.2
FuncScr: afgifte  1.1
FuncAdm: beheer   1
FuncScr: distributie  1.1.1
```

FuncPst: verzending 1.1.2

Gewenst is echter een hiërarchisch overzicht van deze functies, waarbij de aanduiding 'Func' + code niet wordt geschreven. Hiervoor kan het volgende script gebruikt worden:

```
line_open functies.abc          (* 1 *)
sort on                         (* 2 *)
sort 24 80                      (* X *)
sort hierarchie                 (* Y *)

while set $regel line           (* 3 *)
    print_part $regel 10        (* 4 *)

sort off                        (* 5 *)
```

De werking van (1), (2), (3), (4) en (5) is hetzelfde als bij de voorgaande voorbeelden.
Toegevoegd zijn de opdrachten (X) en (Y). Gesorteerd wordt vanaf positie 24, aangezien de regels uit de invoerfile geschreven worden vanaf positie 10 en derhalve de hiërarchische nummering in de sorteerfile begint op positie 24 (34 minus 10). Bovendien wordt nu niet alfabetisch maar hiërarchisch gesorteerd (Y).

De uitvoer ziet er in dit geval als volgt uit:

```
beheer      1
afgifte     1.1
distributie 1.1.1
verzending  1.1.2
aanname     1.2
```

Om in een later stadium het sorteren weer alfabetisch te laten gebeuren, dient sort opnieuw ON gezet te worden en moet de opdracht 'sort hierarchie' worden weggelaten. Met het script:

```
line_open functies.abc          (* 1 *)
sort on                         (* 2 *)
sort 10 80                      (* X *)
<geen sort hierarchie>          (* Y *)

while set $regel line           (* 3 *)
    print $regel                (* 4 *)
```

```
sort off                    (* 5 *)
```

wordt zodoende de file 'functies.abc' alfabetisch gesorteerd (Y) vanaf positie 10 (X; de aanduidingen voor de functienamen worden bij het sorteren genegeerd).
Aangezien de file al alfabetisch op Functie-naam gesorteerd was, zal de uitvoer van dit script hetzelfde zijn als de invoer.

Zie ook: **LINE, LINE_OPEN, PRINT_NUMBER, PRINT_PART.**

SPACES

Syntax: SPACES <aantal | $aantal>

Functie: schrijft het opgegeven aantal spaties naar het current uitvoerkanaal, zonder afsluitende newline. (Lay Out)

Werking: Het keyword **SPACES** heeft als resultaat dat het opgegeven aantal spaties wordt geschreven naar het current uitvoerkanaal. Hierbij wordt géén newline gegeven, ook niet als **NEWLINE** ON staat.
Voorbeeld:

```
print Datum ;
spaces 1
print : ;
spaces 2
date
```

heeft als mogelijk resultaat:

```
Datum :  10-05-1991
```

Om informatie in kolommen te schrijven ligt het **COLUMN**-commando echter meer voor de hand.

Zie ook: **COLUMN, INDENT.**

SUBSTRING

Syntax: SUBSTRING <waarde1|$var1> <waarde2|$var2>

Functie: bekijkt of het eerste argument een substring is van het tweede. Indien dat zo is, wordt de eerste positie van de substring binnen de hoofdstring geschreven. (VAR)

Werking: Met het **SUBSTRING**-commando wordt gekeken of een tekststring voorkomt als gedeelte van een andere tekst-string. Indien dit het geval is, wordt de positie van de hoofdstring geschreven waarop de substring binnen die hoofdstring begint. Is de tekststring geen onderdeel van de andere string, dan wordt '0' (nul) geschreven.

Derhalve hebben de volgende commando's:

```
substring uter computer
substring comp computer
substring scherm computer
```

respectievelijk de volgende resultaten:

```
5
1
0
```

In combinatie met **SET** kan het resultaat van **SUBSTRING** aan een variabele worden toegekend. De hiervoorgenoemde resultaten kunnen bijvoorbeeld aan een variabele $substpos worden toegekend door gebruik te maken van de volgende constructies:

```
set $substpos substring uter computer
set $substpos substring comp computer
set $substpos substring scherm computer
```

Omdat het keyword **SUBSTRING** altijd een uitkomst heeft (0 of een positienummer), is het **niet zinvol** het keyword te gebruiken in combinatie met **IF** of **WHILE**. Indien het gewenst is om slechts te controleren òf een string voorkomt als deel

van een andere string zal derhalve een constructie als de volgende gebruikt dienen te worden:

```
set $substpos substring $zoek $tekst
if equal $substpos 0
    print $zoek komt niet voor in $tekst
else
    print $zoek komt wel voor in $tekst
```

Zie ook: **SET, IF, NOT, WHILE, PRINT_PART.**

SUBTRACT

Syntax: SUBTRACT <$varnaam> <waarde | $waarde>

Functie: vermindert de gegeven variabele met de gegeven waarde. (VAR)

Werking: Met **SUBTRACT** wordt de rekenkundige bewerking aftrekken uitgevoerd. Daarbij wordt uitgegaan van integers, dat wil zeggen gehele getallen (...,-2, -1, 0, 1, 2, ...).
De variabele dient vooraf met het **SET**-commando te zijn geïnitialiseerd. Is dat niet gebeurd, dan volgt een foutmelding en wordt de verwerking van het script afgebroken.
Indien de variabele een waarde heeft of krijgt die geen integer is (bijvoorbeeld een woord of een getal met decimalen), dan wordt het integer-gedeelte van de betreffende variabele gebruikt. Dat wil zeggen dat het gedeelte achter de komma wordt genegeerd. Er vindt derhalve **geen** afronding plaats.
Hetzelfde gebeurt met de waarde die als tweede parameter wordt meegegeven.
De opdrachten:

```
subtract $x 3       (x = computer)
subtract $x -3      (x = 2,1)
subtract $x 3,9     (x = 2)
subtract $x 3,9     (x = 3,1)
```

hebben derhalve als resultaat:

```
x = -3              (0 - 3)
x = 5               (2 - -3)
x = -1              (2 - 3)
x = 0               (3 - 3)
```

Het is ook mogelijk als tweede waarde een variabele mee te geven. Het eerste argument **moet** een variabele zijn. Is dat niet zo, dan wordt de verwerking van het script afgebroken.

Zie ook: **ADD, DIVIDE, MULTIPLY, SET.**

SYSTEM

Syntax: SYSTEM <MS-DOS-commando>

Functie: voert het opgegeven MS-DOS-commando uit. (-)

Werking: Het is mogelijk om tijdens een SDWrite-sessie of vanuit een SDWrite-script een MS-DOS-commando uit te voeren.
Dit kunnen de eigenlijke MS-DOS-commando's zijn, zoals DIR, COPY en CHDIR:

```
system dir a:
system copy *.dat b:
system chdir sdw
```

maar ook aanroepen van andere programma's.

Nadat het aangeroepen commando is uitgevoerd (of het aangeroepen programma is beëindigd), wordt automatisch teruggekeerd naar SDWrite.

Aanroepen met **SYSTEM** worden echter alleen uitgevoerd indien er voldoende geheugenruimte vrij is om MS-DOS te kunnen benaderen.

★ Het commando **SYSTEM** kan uitsluitend worden gebruikt vanuit de commandline-versie van SDWrite.

Zie ook: **DATE, TIME.**

TIME

Syntax: TIME

Functie: schrijft de tijd naar het current uitvoerkanaal of naar een variabele. (-)

Werking: Het commando **TIME** doet niets anders dan het naar het current uitvoerkanaal of een variabele schrijven van de tijd in het formaat uu:mm.
Het mag niet verward worden met het MS-DOS-commando TIME, waarmee ook een nieuwe tijd kan worden opgegeven.

Het is mogelijk de tijd als waarde aan een variabele toe te kennen.
Dit gaat als volgt:

```
set $tijd time
```

De opdracht:

```
print $tijd
```

zal nu dezelfde uitvoer opleveren als **TIME**.

Om de systeem-tijd af te drukken kan tenslotte ook gebruik gemaakt worden van de **standaard SDWrite-variabelen** $SDWhour, $SDWminute en $SDWsecond.
Deze variabelen worden automatisch door SDWrite gevuld.
Het commando:

```
print $SDWhour\:$SDWminute\:$SDWsecond
```

zal de tijd schrijven in het formaat uu:mm:ss.

Zie ook: **DATE, PRINT, SET.**

WHILE

Syntax: WHILE <voorwaarde> <actie>

Functie: voert steeds opnieuw de gegeven actie uit zolang aan de gegeven voorwaarde wordt voldaan. (PRG)

Werking: Met een **WHILE**-constructie wordt het uitvoeren van een bepaalde actie (een commando of een reeks commando's tussen **BEGIN** en **END**) gebonden aan een bepaalde voorwaarde. Zolang aan de betreffende voorwaarde wordt voldaan, wordt de betreffende actie uitgevoerd; wordt niet (langer) aan de voorwaarde voldaan, dan wordt de betreffende actie niet (langer) uitgevoerd.

De actie in een **WHILE**-constructie kan bestaan uit één commando of uit een reeks commando's tussen **BEGIN** en **END**. Een voorbeeld:

```
newline off
display Doorgaan? (j/n)          (* 1 *)
set $antwoord prompt
newline on

while NOT equal $antwoord n      (* X *)
    BEGIN
    <* serie opdrachten *>       (* 2 *)
    newline off
    display Doorgaan? (j/n)      (* 3 *)
    set $antwoord prompt
    newline on
    END
print ... user-interrupt ...     (* 4 *)
print End of run
```

In dit voorbeeld wordt de gebruiker allereerst gevraagd of moeten worden doorgegaan of niet (1). Toetst de gebruiker hier in dat niet moet worden doorgegaan ('n'), dan wordt niet voldaan aan de voorwaarde in de **WHILE**-constructie (X), en zal de slottekst (4) naar het current uitvoerkanaal geschreven worden.

Indien aan het begin (1) een ander antwoord dan 'n' wordt gegeven, wordt wel aan de voorwaarde in de **WHILE**-constructie voldaan (X), en zullen de opdrachten (2) worden uitgevoerd gevolgd door een herhaling van de vraag of moet worden doorgegaan (3). Vervolgens zal worden teruggekeerd naar het begin van de **WHILE**-constructie (X).

Indien bij (3) nu wèl 'n' is geantwoord, wordt op dat moment niet langer voldaan aan de voorwaarde, en zal de slottekst worden geschreven (4). Indien opnieuw iets anders dan 'n' is geantwoord, zullen opnieuw de opdrachten onder (2) worden uitgevoerd, gevolgd door een herhaling van de vraag (3).

De commando's achter **WHILE** kunnen op deze wijze een onbepaald aantal maal worden herhaald. Het is alleen mogelijk deze cyclus te verlaten door bij een van de herhalingen van vraag (3) te antwoorden met 'n'.

Een veel voorkomende fout met een **WHILE**-constructie ontstaat wanneer deze zo is opgezet dat **altijd** aan de voorwaarde wordt voldaan, en er derhalve aan de herhaling van de **WHILE**-actie geen einde komt.

Zo'n zogenaamde oneindige lus ontstaat bijvoorbeeld indien in het voorgaande voorbeeld de herhaling van de vraag (3) wordt weggelaten:

```
newline off
display Doorgaan? (j/n)          (* 1 *)
set $antwoord prompt
newline on

while NOT equal $antwoord n      (* X *)
    BEGIN
    <* serie opdrachten *>       (* 2 *)
    <* geen vraag *>             (* 3 *)
    END

print ... user-interrupt ...     (* 4 *)
print End of run
```

In dit geval zal het script nooit meer uit de while-constructie raken indien de eerste maal (1) een ander antwoord dan 'n'

gegeven wordt. In dat geval wordt namelijk aan de **WHILE**-voorwaarde voldaan (X), zodat de serie opdrachten (2) wordt uitgevoerd, waarna direct weer wordt teruggekeerd naar het begin van de **WHILE**-constructie **zonder dat een nieuwe waarde voor $antwoord wordt ingelezen**. Er wordt derhalve opnieuw aan de **WHILE**-voorwaarde voldaan (X), de serie opdrachten (2) wordt opnieuw uitgevoerd, enz. Dit eindigt pas als de gebruiker het script afbreekt (bijvoorbeeld met Ctrl_Alt_Del).

Zie ook: **BEGIN**, **END**, **IF**.