Een filter voor de Calcomp elektrostatische plotter

Een filter voor de Calcomp elektrostatische plotter.

Geschreven door Ron Kersic en Jeroen Voeten in het kader van een stage aan het TUE-rekencentrum.

Stagebegeleider: C. Braam
CC-Note 47

Inhoudsopgave

0. Inleiding

Omwille van het maken van plaatjes kunnen files met plotgegevens aan de Calcomp 5835XP-plotter worden aangeboden. Helaas voldoen lang niet alle files aan de door Calcomp gedicteerde regels. Zulke files kunnen het 'hangen' van de plotter veroorzaken, waarna enkel een herstart nog uitkomst biedt.
Het ware gewenst dat alleen die files naar de plotter worden verzonden die de plotter in generlei wijze van zijn stuk kunnen brengen, zodat de correcte werking van de plotter op ieder moment gegarandeerd kan worden. Hiertoe werd een zogeheten filter geschreven dat enkel de correct opgebouwde files naar de plotter doorzendt. De filterende werking van het programma was echter niet van dien aard dat het 'hangen' van de plotter definitief tot het verleden behoorde.
De beschrijving van de constructie van een filter dat de pretentie heeft enkel correcte files door te laten, is het onderwerp van dit verslag.

1. Het dataformaat

De Calcomp plotter wordt bestuurd door een zogeheten controller. Alle data, die naar de plotter wordt gezonden, passeert eerst deze controller. Enkel en alleen wanneer de controller de data correct heeft bevonden, wordt deze doorgesluisd naar de plotter zelf.
Een plotfile bestaat uit een of meerdere opeenvolgende datamessages. Een datamessage is de eenheid van informatie- transport naar de controller. In plaats van de informatie byte voor byte naar de controller te versturen, wordt deze gebundeld in pakketjes: de datamessages. Elke datamessage, ongeacht de informatie die erin vervat ligt, is van de onderstaande vorm:

<sync seq.> <data sequence> <EOM> <suffix> <resp. req.>

waarbij

<data sequence> = <bias> <data>

De termen in dit diagram zijn dermate belangrijk en immer wederkerend dat ze een gedegen uitleg erdienen.

sync seq. (sync sequence)
De sync sequence markeert het begin van een datamessage. De sync sequence bestaat uit één of twee ASCII karakters met een ordinaal waarde tussen 0 en 127. Wanneer de sync sequence uit twee tekens wordt gevormd (hetgeen door Calcomp aanbevolen wordt) dienen deze tekens identiek te zijn. Helaas wordt deze conventie lang niet altijd gevolgd, hetgeen onherroepelijk tot problemen leidt.

bias
Er bestaan verschillende stelsels om karakters te representeren en te ordenen. Hiervan is ASCII ongetwijfeld de meest gebruikte. Het bias-karakter is het eerst afdrukbare teken binnen het gebruikte stelsel. In het ASCII-stelsel is het eerst afdrukbare karakter de spatie welke ordinaal getal 32 heeft, en alle voorafgaande karakter zijn besturingskarakters (zoals de linefeed).
Het bias-karakter behoort direkt op de sync sequence te volgen, en is het eerste karakter van de data sequence.

data
In dit gedeelte is de eigenlijke informatie vervat. De structuur ervan wordt later in detail behandeld.
Sommige protocollen maken gebruik van een **checksum** om de integriteit van de data te garanderen. Deze checksum is laatste element van het data gedeelte (dus nog net voor de EOM). De huidige instelling van de plotter voorziet niet in een checksum.
Bij alle informatie in het data gedeelte, de checksum uitgezonderd, is de bias opgeteld, zodat deze informatie binnen het gebruikte karakterstelsel valt.

karakter moet, onder de tekens die naar de controller worden gestuurd, uniek zijn en moet een ordinale waarde hebben die tussen 0 en 1F hexadecimaal ligt, waarbij bovendien het EOM-karakter niet gelijk mag zijn aan de bias.
Hoewel de Calcomp documentatie er niet over rept, nemen wij aan dat het EOM-karakter niet gelijk is aan het karakter waaruit de sync sequence wordt gevormd. Deze alleszins redelijke aanname voorkomt ambiguïteiten.

Res. Req. (Response request string)
De response request string bestaat uit 1 tot 15 karakters. Zoals de sync sequence het begin van een datamessage aanduidt, geeft de response request string het einde weer. Indien deze string ontbreekt in een datamessage, zal de plotter gaan 'hangen'. Gezien het grote aantal files waarin de response request string niet voorkomt, is het niet verwonderlijk dat het blokkeren van de plotter een dagelijkse realiteit is geworden.

Suffix (End of message suffix)
Het gedeelte begrensd door het EOM-karakter en de response request string heet de suffix (of voluit de end of message suffix). De tekens in de suffix worden op geen enkele wijze door de controller geïnspecteerd dan wel geïnterpreteerd. Alle afdrukbare ASCII-karakters worden simpelweg naar het het scherm van de plotter-terminal gestuurd. De suffix speels geen enkele rol van belang bij het plotproces, maar kan gebruikt worden om informatie aan de operateur door te geven.

Goede en gedetailleerde informatie over datamessages en hun opbouw is terug te vinden in hoofdstuk 2 van het **Calcomp online reference manual.**

2. De header

Zoals al eerder gezegd bestaat een plotfile uit meerdere datamessages. De eerste datamessage heet de header. Deze header bevat informatie omtrent de interne parameters van de controller, die van invloed zijn op het plotproces. In de header is dus geen daadwerkelijke plotdata voor een plaatje aanwezig. Enkele zeer belangrijke en noodzakelijke parameters die met behulp van instructies in de header gezet worden, zijn:

Search adress

Iedere header moet voorzien zijn van een search adress met waarde 001 (let wel: 001 en niet 1), en wel meteen volgend op de bias. Zulk een search adress initialiseert de plotter.

Radix

De radix vormt de basis van het door de controller gebruikte talstelsel en moet dan ook persé gedefinieerd worden. De radix hangt af van de in de plotfile gebruikte afdrukbare karakters. Zij c het maximum van de ordinale waarden van de in de plotfile gebruikte afdrukbare karakters, dan geldt

$$\text{radix} = c - \text{bias} + 1$$

Ondanks het onmiskenbare belang van de radix is het aantal files dat niet van een radixdefinitie vergezeld gaat hoog. Indien een radixdefinitie ontbreekt definieert het ontwikkelde filter een standaard radix van 65 (die ook gehanteerd wordt bij het initialiseren van de controller). Aangezien de radix het talstelsel van de controller bepaalt, kan het meegeven van een standaard radix leiden tot uit hun verband gerukte plaatjes.

Escape sequence

In de zogeheten escape sequence worden een aantal belangrijke parameters van de controller gedefinieerd. Van het grote aantal in te stellen parameters zijn er twee van groot belang:

Buffer

Ieder datamessage die naar de controller wordt verstuurd, wordt opgeslagen in een input buffer. Het aantal buffers en hun grootte moet worden bepaald met behulp van de buffer escape sequence.

Response request string

De al eerder genoemde response request string wordt niet hardware-matig door de controller opgelegd, doch moet in een escape sequence vastgelegd worden. De aldus gedefinieerde string wordt in elke datamessage (de header incluis) gebruikt.

Tot slot kunnen in een escape sequence parameters vastgelegd worden die voor de communicatie met de controller van belang zijn. Omdat die voor een begrip van de werking van de plotter er niet toe doen, wordt er hier niet nader op ingegaan.

Om te kunnen waarborgen dat ieder plaatje vanuit dezelfde uitgangspositie begint, kunnen in de header, volgend op de escape sequence, een aantal opdrachten worden opgenomen die een aantal instelmogelijkheden van de plotter (en niet de controller) van een standaardwaarde voorzien. **Select pen 1** en **Set scaling to 1**, die een standaard (virtuele) pen en schaalgrootte kiezen, zijn het meest voor de hand liggend.

Meer informatie omtrent de header is terug te vinden in hoofdstuk 2 van het **Calcomp online reference manual.**

3. De plotmessage

Na de header volgen nul of meer plotmessages. Plotmessages zijn datamessages die informatie voor de plotter (en dus niet voor de controller) bevatten. De data sequence van een plotmessage kan een groot aantal instructies voor de plotter bevatten. Daar de Calcomp documentatie bepaald niet volledig en consistent is, is in **Appendix A** een compleet overzicht van de beschikbaare instructies opgenomen.
De laatste plotmessage in een file behoort als laatste instructie een **end-of-plot** te bevatten, die de plotter opdracht geeft tot het rasteren en zichtbaar maken van de ontvangen plotdata.

4. Het filter

Het ontwikkelde filter pluist een aangeboden file na op inconsistenties, onvolkomenheden en fouten die een correcte werking van de plotter in de weg staan. Wanneer mogelijk zal het filter trachten de tekortkomingen aan te vullen dan wel te verbeteren, zodanig dat een goede plot wél mogelijk is. In het geval van een onherstelbare fout zal het filter in ieder geval een passende foutmelding genereren omtrent de aard van het euvel.

4.0 Het filteren van een header

De eerste taak van het filter is het filteren van de header. Het filter is voor de header veel meer vergevingsgezind dan voor plotmessages: een tekortkoming wordt altijd omzeild door terug te grijpen op het gebruik van standaardwaarden. Enkel van de sync sequence en de bias wordt de aanwezigheid geëist. Een EOM, bufferdefinitie of response request string worden, indien nodig, op hun defaultwaarden geïnitialiseerd. Daar niet bestaande instructies domweg worden genegeerd, moet een header wel erg barok zijn wil hij door het filter verworpen worden.
Het filteren van een header volgt de structuur van een header:

sync sequence

Voor alles gaat het filter op zoek naar een sync sequence die het begin van de header markeert. Omdat vrijwel geen enkele, door ons bekeken plotfile voldeed aan de door Calcomp opgelegde regels restte ons niets anders dan de premisse dat de eerste karakters van een plotfile de sync sequence voorstellen. Met een eenvoudige case-analysis wordt de sync sequence bepaald en opgeslagen voor later gebruik (de plotmessages beginnen immers ook met deze sync sequence).

bias

Het filter verwacht direct na de sync sequence het bias-karakter. Omtrent de waarde van dit karakter maakt het filter geen veronderstellingen: de bias wordt niet geïnspecteerd, maar enkel opgeslagen voor later gebruik.

instructies

Rest nog het verwerken van de eigenlijke instructies. Hiertoe haalt het filter de eerst volgende waarde op en trekt er de bias van af (alle instructies -inclusief hun parameters- zijn immers vermeerderd met de bias) waarna de eigenlijke opcode overblijft. Wanneer deze opcode overeenkomt met een daadwerkelijke instructie, worden de eventuele parameters van deze instructie opgehaald, gecontroleerd op hun consistentie en correctheid, en bewaard voor gebruik bij het creëren van de nieuwe

header. Een niet herkende opcode wordt genegeerd. Het bovenstaande wordt net zolang herhaald totdat één der onderstaande gevallen zich voordoet:

0. De volgende sync sequence wordt gevonden. Het EOM-karakter is dan niet aanwezig in deze header.

1. Het EOM-karakter wordt gevonden. Omdat de plotter geen hardwarematig opgelegd EOM-teken kent, wordt ieder karakter, kleiner dan de bias en niet gelijk aan het karakter waaruit de sync sequence wordt gevormd, als het EOM-karakter beschouwd.

2. Het einde van de file is bereikt. Er kan in dit geval geen enkele uitspraak worden gedaan over de lengte van de header en het begin van het plaatje. Het plaatje wordt dan ook verworpen.

4.1 De creatie van een header

Uit de, zoals boven beschreven, verzamelde informatie wordt nu een header gedestilleerd die volledig is en geheel aan de Calcomp conventies voldoet. Evenals het inlezen van headerinformatie, volgt het creëren van de nieuwe header de headerstructuur.

sync sequence

In plaats van de ingelezen sync sequence wordt de standaard sync sequence (i.e. zoals die op de plotter is ingesteld) weggeschreven. Deze sequence wordt gevormd door het ASCII-karakter met ordinaal getal 2.

bias

De ingelezen bias wordt eenvoudig overgeheveld naar de nieuwe header.

instructies

Alle verzamelde headerinstructies worden, daar waar nodig aangevuld met standaardwaarden, in de juiste volgorde en in de juiste vorm weggeschreven.
Volgend op de laatste instructie wordt het EOM-teken, met ordinaal getal 3, weggeschreven (een eventuele checksum wordt genegeerd).

suffix

Na het inlezen van de headerinstructies doet zich één van de boven vermelde gevallen 0 en 1 voor:

<sync seq.> <data sequence> <EOM> <suffix> <resp. req.>

waarbij

<data sequence> = <bias> <data>

0 Er komt geen EOM-teken in de te filteren header voor en derhalve kan er geen sprake zijn van een suffix (zie bovenstaand plaatje). Dan wordt er een standaardsuffix ("Plotting begins...") gevolg door de response request (daadwerkelijk gedefinieerd of standaard) aan de gefilterde header-in-wording toegevoegd.

1. Er is een (eventueel lege) suffix aanwezig, wellicht gevolg door een response request string. Wanneer in de oorspronkelijke, te filteren header zulk een response request string werd gedefinieerd, dan wordt met een eenvoudig pattern match-algoritme getracht deze string op te sporen. Nu kunnen zich twee gevallen voordoen:

0. De response request string werd gevonden. Alle tekens tussen het EOM-teken en deze string worden simpelweg van de ene header naar de andere overgeheveld, evenals de response request string.

1. De response request string werd niet gevonden. Alle voorgaande tekens, vanaf het EOM-karakter worden ook nu domweg gecopieerd, waarna de gedefinieerde response request string

wordt weggeschreven.

Wanneer in de oorspronkelijke header geen response request string werd gedefinieerd, is deze leeg. Alle tekens tussen het EOM-teken een EOF danwel sync sequence worden gecopieerd.

4.2 Het filteren en creëren van een plotmessage

Na de header volgen de plotmessages. Deze messages worden gefilterd en gemodificeerd volgens het onderstaande stramien:

sync sequence

In tegenstelling tot de header, hoeft er nu niet meer bepaald te worden met welke sync sequence we van doen hebben. Het filter gaat enkel op zoek naar de eerst volgende sync sequence, zoals die uit de header bekend is. Wordt zulk een sequence niet gevonden, dan was er blijkbaar geen plotmessage meer en het filterproces mag als beëindigd worden beschouwd. Wanneer deze sequence wel wordt gevonden dan wordt de standaard sync sequence weggeschreven.

bias

Het teken volgend op de sync sequence moet de bias zijn (niet persé dezelfde als in de header gebruikt werd) en wordt meteen weggeschreven.

instructies

Het verwerken van de instructies verloopt volgens het volgende standaard recept:

0. De eerstvolgende byte wordt ingelezen en verminderd met de bias.

1. Komt de zo verkregen waarde voor als opcode van een instructie, dan wordt de bijbehorende parse-functie aangeroepen, die de eventuele parameters van deze instructie inleest. Wanneer deze parameters correct zijn wordt de instructie, natuurlijk inclusief de parameters, vermeerderd met de waarde van de bias weggeschreven.

2. Komt de verkregen waarde niet voor als een opcode van een bekende instructie, dan zou deze waarde plus

5. Enkele bijzonderheden van de Calcomp instructies

Instructieformaat

Een aantal van de beschikbare instructies is te herleiden tot de vorm

<opcode> <parameter> ... <parameter>

Bij zulk een instructie hoort een parse-functie die bij het inlezen van <opcode> aangeroepen wordt en de parameters ophaalt en verifieerd.
Een aantal instructies is echter van de vorm

<opcode> ... <opcode> <parameter> ... <parameter>

waarbij de eerste opcode als het ware een hele klasse van instructies aanduidt. Vandaar dat de parse-functie een andere parse-functie kan aanroepen, net zolang totdat enkel nog de parameters (voorzover aanwezig: sommige instructies hebben geen parameters en hebben zodoende geen parse-functie nodig) overblijven.

Delta instructies

Een wel zeer belangrijke klasse van instructies wordt gevormd door de zogeheten **delta instructies**. Deze instructies bewerkstelligen op zeer flexibele wijze het bewegen van de (virtuele) pen van de plotters.
Een algemene vorm voor een delta instructie is een opcode, gevolgd door resp. 0 to 3 x-parameters en 0 tot 3 y-parameters, afhankelijk van de opcode en waarbij minstens één parameter meegegeven dient te worden. Een delta instructie definieert een vector (X,Y), die bepaald wordt door de x- en y-parameters. De x-parameters stellen tezamen een radix-tallig getal, evenals de y-parameters. Zoals 123 in het decimale stelsel eigenlijk 1*10*10 + 2*10 + 3 voorstelt, zo stelt 123 in het radix-tallig stelsel 1*radix*radix + 2*radix + 3 voor. Hier duikt het enorme belang van de radix op. Een verkeerde radix betekent zonder meer een totaal verwrongen plaatje.
De absolute waarden van de X- alsook de Y-component dienen onder een vaste waarde te blijven (veelal 32767). Indien een delta instructie een vector definieert die hier niet aan voldoet, dan moet deze vector opgesplitst worden in meerdere vectoren die wel onder de gegeven bovengrens blijven. Dit resulteert in meerdere delta instructies. Echter, een aantal instructies, bijvoorbeeld de cirkel-instructie, maken gebruik van delta instructies. Omdat de delta instructies nu als parameter fungeren is het splitsen niet geoorloofd (want dan zou het aantal parameters toenemen).

Een complete beschrijving van de delta instructies is terug te vinden in hoofdstuk 3 (**pen control and plotting**) van de Calcomp handleiding.

Plot copies

De plot copies instructie geeft de plotter opdracht om een en hetzelfde plaatje meerdere malen af te drukken. Deze instructie mag

de bias wel eens de checksum kunnen zijn. Dit is te achterhalen door de volgende byte (mits aanwezig) op te halen. Wanneer deze gelijk is aan het EOM-karakter, dan hebben we inderdaad de checksum te pakken, die prompt genegeerd wordt.
Wanneer het niet de checksum kan zijn (i.e. het erop volgende teken is niet het EOM-teken) dan hebben we een fout gevonden. De waarde stelt blijkbaar een onbekende opcode voor. De enige veilige aktie is in dit geval abortie van het filterproces.

Dit proces wordt net zolang herhaalt, totdat er of een EOM of een sync sequence of een EOF opduikt. In al deze gevallen kan de datasequence als afgesloten worden beschouwd.

De rest

Volgend op het succesvol verwerken van de laatste instructie in de data sequence, worden achtereenvolgens het EOM-teken en de in de header gedefinieerde response request string weggeschreven. Een eventuele suffix wordt, gezien zijn beperkte waarde, genegeerd.

enkel en alleen als eerste instructie in de eerste plotmessage voorkomen. Vandaar dat het verwerken van de instructies in het filter vooraf wordt gegaan door een test op de aanwezigheid van de plot copies instructie.

End-of-plot

De end-of-plot instructie geeft de plotter opdracht tot het rasteren en afdrukken van de ontvangen plotdata. Deze instructie mag alleen als laatste instructie in de laatste plotmessage voorkomen omdat iedere instructie, volgend op een end-of-plot, door de controller genegeerd wordt.

6. Enkele opmerkingen over het programma

Berichten

Het programma kan een aantal waarschuwingen, foutmeldingen en mededelingen produceren. Een compleet overzicht van deze berichten is terug te vinden in **Appendix B.**

Structuur

De structuur van het programma is zoveel mogelijk conform de in dit geschrift beschreven opbouw.
Echter, het programma is lijvig en vele ontwerpbeslissingen, die eerst zeer zinvol leken, bleken achteraf niet zo gelukkig gekozen. Gezien de beperkte tijd en de wens een degelijk, functionerend programma op te leveren is een aantal programma-technische lelijkheden (in gedachte houdend Dijkstra's "Beauty is our bussiness...") niet verholpen. Dankzij de consequent volgehouden structuur van het programma moeten deze met relatief gemak te verhelpen zijn.

7. In de toekomst te verwezenlijken mogelijkheden

Bufferlengte

Zoals al eerder gezegd, de lengte van een data sequence mag de grootte van de gealloceerde buffer niet overschrijden. Wanneer het EOM-teken niet meer in de buffer past, leidt dit tot een 'hangen' van de plotter, zoals wij op pijnlijke wijze ondervonden hebben. De remedie hiervoor is om tijdens het creëren van gefilterde datamessages de lengte van de datasequences goed in het oog te behouden en, wanneer nodig, deze af te kappen. De door ons geïmplementeerde oplossing voldoet hier wel aan, maar is nogal rigoreus in het afkappen. Het afkappen gebeurt op grond van de maximaal mogelijke lengte van enige instructie, in plaats van per instructie de lengte in ogenschouw te nemen. Wij zijn ons bewust van de lelijkheid van deze oplossing, maar tijdgebrek noopte tot het achterwege laten van een elegantere oplossing.

De lengte van een plot

Een van de wensen van het Rekencentrum is dat, na het afdrukken van een plaatje, de voor deze plot verbruikte hoeveelheid papier bekend is. Een oplossing, niet door ons geïmplementeerd, lijkt mogelijk. Immers: iedere beweging van de virtuele pen wordt bewerkstelligd met de al eerdere genoemde delta instructies. Hieruit moet de totale afgelegde afstand in de X-richting (i.e. de verbruikte hoeveelheid papier) te bepalen zijn.

Opsplitsten in deelprogramma's

Het filterprogramma is nogal omvangrijk. De lees- en handelbaarheid van het programma is gebaat bij een opsplitsen in meerdere deelprogramma's.

8. Conclusies en opmerkingen

- De functie van het filter zou eigenlijk door de controller van de plotter vervuld moeten worden.
- De plotter kan blijven 'hangen' als gevolg van slechte invoerdata. Dit moet zonder meer als een ernstige ontwerpfout van de Calcomp-ontwerpers worden aangemerkt.

- De uitvoer van grafische pakketten voldoen bedroevend slecht aan de plottertaal specificaties.

- Het ontworpen filter is zo van opzet, dat wijzingen en verdere uitbreiding zonder veel omhaal gerealiseerd kunnen worden.

9. Enkele problemen ter afsluiting

Zij X de collectie files. Laat met F: X -> X de filterende werking van het programma weergegeven zijn, d.w.z. voor iedere x in X is F.x ="de gefilterde versie van file x".
Weerleg of bewijs de volgende uitspraak:

$$(\underline{A}_x: x \text{ in } X: F.x = F^2.x)$$

waarbij $F^2.x = F.(F.x)$ voor alle x in X

Bovenstaand probleem inspireerde onze stagebegeleider tot de onderstaande interessante problemen:

Zij X de verzameling files en laat C $\subseteq$ X de collectie files zijn die aan de door Calcomp gedefiniëerde syntax en semantiek voldoen. Zij P tot slot de verzameling plaatjes, terwijl F: X -> X de filterfunctie is die door het programma wordt gerealiseerd en G: C -> P de plotfunctie, d.w.z de functie die van een plotbare file een plaatje maakt.
Bewijs dat

1. x in X => F.x in C
2. x in C => G.(F.x) = G.x

10. Dankzegging

Wij willen een woord van dank richten aan de medewerkers van het Rekencentrum die ons, meer dan eens, met raad en daad terzijde hebben gestaan.
Meer dan erkentelijk zijn wij onze stagebegeleider C. Braam. Zonder zijn, nooit van logica en waarheid gespeende, kritiek en opmerkingen zou deze stage voor ons nooit zo vruchtbaar en waardevol geworden zijn.

Appendix A
De Calcomp instructies

Instructie	Opcode in Hex	pagina
no operation	00	8-2
search adress	01	3-2*
pen down	02	3-2*
pen up	03	3-3
pen select	04	3-4
symbol string count	05	5-6
controller symbol scale	06	5-10
radix	07	3-3*
disable double-buff	08 00	3-6*
enable double-buff	08 01	3-7*
set response suffix...	08 02	3-29*
set turnaround delay	08 03	3-21*
set good response	08 04	3-15*
set bad response	08 05	3-16*
set response request	08 06	3-11*
set max waiting time	08 07	3-22*
set waiting response	08 08	3-23*
disable <DEL>	08 09	3-5*
set 16 buffers	08 0A	3-8*
set 8 buffers	08 0B	3-8*
set 4 buffers	08 0C	3-8*
set 2 buffers	08 0D	3-8*
scaling	09	4-15
pause	0A	8-5
dashline	0D	7-6
end of plot	0F	3-30*
delta	10- 3F	3-14
pass through...	0B	8-2
manual	0B 00 03	8-4
direct pass nop	0B 00 04	8-3
plotter performance	0B 00 05	3-8
newplot	0B 00 06	8-4
chordal tolerance	0B 00 08	6-2
font select	0B 00 09	5-2
symbol characteristics	0B 05 0E	5-16
plotter symbol scaling	0B 07 06	5-3
dash bypass	0B 08	7-2
circle and arcs	0C	6-7

Instructie	Opcode in Hex	Pagina
plotting symbol from...	0E	5-32
operator message	0E 06	8-6
operator message with...	0E 07	8-7
select symbol set 1	0E 09	5-27
select symbol set 2	0E 0A	5-29
select symbol set 3	0E 0B	5-31
select symbol set 4	0E 0C	5-31
plotting symbol from...	0E 0F	5-33
user-defined symbol	0E sset 0E 0D	5-34
erase user symbol set	0E sset 0E 0E	5-41
negate	0E 11	9-6
set pen	0E 12	9-3
paper cutter	0E 13 00	9-15
top of form	0E 13 01	9-16
pattern fill	0E 15	9-7
hatch	0E 15	9-10
set pat	0E 16	9-11
start plot	0E 18	9-16
force plot	0E 19	9-16
plotter select	0E 1A	9-2
plot copies	0E 1B	9-2
mirror	0E 1D	4-7
window	0E 1E	4-10
rotation	0E 1F	4-2
new pen	0E 20	11-2
pattern fill	0E 21	11-12
set pat	0E 22	11-51
color modify	0E 23	11-37
color sequence	0E 24	11-10
diskIO	0E 25	11-76
new level	0E 26	11-79
xset pen	0E 28	11-46
set level	0E 29	11-77
raster fill	0E 2A	11-85
pixel	0E 2B	11-88
area fill	0E 2C	11-25

Nb: de met een * voorziene paginanummers hebben betrekking op header-instructies en zijn terug te vinden op de geel-gelabelde paginas in de Calcomp handleiding.

Appendix B
Waarschuwingen, foutmeldingen en berichten.

Fouten

In het volgende staan de foutmeldingen die door het filter worden gegenereeerd. Deze meldingen worden enkel en alleen gegenereerd wanneer het het filterproces niet meer voortgezet kan worden.
Enkele foutmeldingen zijn zo vanzelfsprekend dat ze geen nadere uitleg verdienen.

Fouten die kunnen voorkomen in headers

1 Unexpected end of file found in header
De header hield plotsklaps op. Blijkbaar is een stuk van de header weggevallen.

2 Wrong countnumber for message string
Een message string kan slects 30 nibble's (is 15 bytes) lang zijn. Een message string van lengte 0 is eveneens niet correct.

3 inputfile not specified
Het filter moet weten wele plotfile gefilterd dient te worden.

4 <inputfile> was not found or does not exist

5 Getvalue could not read from file
De filterfunctie getvalue faalde bij het lezen van een byte uit de inputfile

6 Putvalue could not write to file

7 outputfile not specified
Het filter moet weten waar het gefilterde resultaat naar toe moet worden geschreven.

8 <outputfile> could not be created.

Fouten die kunnen voorkomen in plotmessages

0 Unbiased value found in data.
De ordinale waarde van iedere data-element moet groter zijn dan of gelijk zijn aan de bias.

1 Wrong data. Bias not found.

2 EOF found in data.

3 Wrong pen selected
De pen select instructie staat slechts pennummers van 1 tot en met 16 toe.

4 Unvalid nmber of characters in symbolstring.

5 Delta too long. Split not allowed.
Een delta instructie reslteerde in een te lange vector. Het splitsen van deze vector is echter niet toegestaan.
Bij cirkel-instructies worden delta-instructies gebruikt als parameters. Het splitsen is niet geoorloofd omdat dan het aantal parameters zou toenemen.

6 Delta command expected in controller command

7 Incorrect segment alignement in dashline-command.

8 Rotation angle too large.

9 Wrong operand in mirror-command.

10 Wrong operand in window-command.

11 Wrong operands in plot symbols-command.

12 Count too high in message.
Een message mag maximaal 64 tekens bevatten.

13 Wrong channel in plotters select.

14 The plot copies-command may only be specified at the beginning of a programm.

15 Too many plots specified
64 is het maximum.

16 Wrong parameters in setpen-command.

17 Wrong imagenumber in negate-command.

18 Wrong rate specified in async-command.

19 Wrong pattern specified in pattern/hatch-command.

20 Wrong parameters specified in setpat-command.

21 Wrong paper-command.

22 Wrong colorpas number.

23 Wrong parameters in xsetpen-command.

24 Wrong patternnumber in xsetpat-command.

25 Wrong parameters in disIO.

26 Wrong parameters in setlevel-command.

27 Delta-command expected in raster fill.

28 Wrong parameters in user-defined symbol.

29 Wrong parameters in plotting symbols-command from selected symbol set.

30 Wrong parameters in font select-command.

31 Wrong parameters in chordal tolerance-command.

32 Unknown direct pass thru-command.

33 Wrong parameters in symbol characteristics-command.

34 Delta-command in plotter symbol scaling expected.

35 Wrong type-parameters in dash bypass-command.

36 Pass thru 8-bits direct to plotter not allowed.
Hoewel deze instructie toegestaan is, is zijn gebruik zo dubieus, dat hij geweigerd wordt.

37 Unknown circle-command.

38 Delta expected in circle- or arc-command.

39 Wrong pixel value found in raster fill-command.

40 Bad command found.

Waarschuwingen

Het optreden van een waarschuwing impliceert niet dat het filter-proces onsuccesvol is. Een waarschuwing maakt de gebruiker erop attent dat het filter maatregelen heeft getroffen om gevonden tekortkomingen te omzeilen.

1 Unknown escape command detected and ignored.
In een escape sequence dook een onbekende instructie op, die genegeerd werd.

2 No EOM detected in header. EOM has been added but a good plot is not guaranteed.
De header bevatte geen EOM, waardoor wellicht informatie in deze header verkeerd geïnterpreteerd wordt.

3 Response request not found and added.

4 EOF found while scanning suffix.

5 No start plot after a force plot.
Een force plot-instructie moet persé gevolgd worden door een start plot-instructie.

Berichten

Het filter houdt met behulp van berichten de gebruiker op de hoogte van de toegepaste correcties.

1 Search adress set to default value.

2 Radix set to default value.

3 Buffer set to default value.

4 Requeststring set to default value.

5 Good response message set to default value.

6 Bad response message set to default value.

7 Waiting response message set to default value.

8 Maximum waiting time set to default value.

9 Turnaround delay set to default value.

10 Response suffixlength set to default value.

11 Pennumber set to default value.

12 Scale factor set to default value.

13 Suffix string set to default value.

Appendix C
De programmatekst in C

```
/*
   parser voor plot-headers.
   Versie 0, dinsdag ergens in juni
*/


#include <stdio.h>

#define FALSE 0
#define TRUE !FALSE
#define SYNC 2
#define EOM 3
#define MAXOPLEN 25


int sync1, sync2;                       /* SYNC-characters */
int bias;                               /* bias-waarde */
int OriginalEOM;                        /* End Of Message-character */
int shigh,smid,slow;                    /* Search adress parameters */
int radix;                              /* De gebruikte radix */
int pennum, scale;                      /* Default pen en schaalfactor */
int geensuffix;                         /* Is er een suffix? */
int buffer;                             /* Het bufferformaat */
int turnaround;                         /* Turnaround delay */
int waittime;                           /* Max waitingtime */
int suffixlength;                       /* Response suffix lengte */
int disable_del;                        /* Disable <DEL> on? */
int requeststring[31];                  /* Request string in nibbles */
int requestcode[16];                    /* Request string */
int requestdefined;                     /* Werd er een request gedef.? */
int goodresponse[31];                   /* Good response string */
int badresponse[31];                    /* Bad response string */
int waitresponse[31];                   /* Waiting response string */
int lastvalue;                          /* Laatst ingelezen waarde */
int buflen;                             /* Lengte van de plotterbuffer */
int inbuf;                              /* Aantal tekens in buffer */
FILE *fp_in;                            /* Invoerfilepointer */
FILE *fp_out;                           /* Uitvoerfilepointer */
char *inputfile;                        /* De te verwerken plotfile */
char *outputfile;                       /* De resultaat van het filter */
char *progname;                         /* De programma-naam */

extern void do_extended();              /* do_extended is verwikkeld
                                           in een wederzijdse recursie */
```

```
void error(num)          /* Produceert een foutmelding en breekt het prog. af */
  int num;
  {
   printf("Fatal header error: ");
```

```
   switch(num)
     {
      case 0  : /* SYNC niet gevonden */
                printf("File not recognized: sync not found.\n");
                break;
      case 1  : /* EOF gevonden midden in de header */
                printf("unexpected end of file found in header.\n");
                break;
      case 2  : /* Verkeerde string-count */
                printf("wrong countnumber for messagestring.\n");
                break;
      case 3  : /* <inputfile> werd niet gespecificeerd */
                printf("inputfile not specified.\n");
                printf("Usage: %s <inputfile> <ouputfile>.\n",progname);
                break;
      case 4  : /* <inputfile> bestaat niet of werd niet gevonden */
                printf("%s was not found or does not exist.\n",inputfile);
                break;
      case 5  : /* Getvalue faalde in het lezen van een waarde */
                printf("Getvalue could not read from file.\n");
                break;
      case 6  : /* Putvalue faalde in het wegschrijven van een waarde */
                printf("Putvalue could not write to file.\n");
                break;
      case 7  : /* <outputfile> werd niet gespecificeerd */
                printf("Outputfile not specified.\n");
                printf("Usage: %s <inputfile> <outputfile>.\n",progname);
                break;
      case 8  : /* <outputfile> bestaat niet of werd niet gevonden */
                printf("%s was not found or does not exist.\n",outputfile);
                break;
     }
   printf("Header creation canceled: sorry\n");
   fclose(fp_out);
   exit(num);
  }



void data_error(num) /* Produceert foutmeldingen omtrent de data */
  int num;
  {
   printf("Fatal data error: ");
   switch(num)
     {
      case 0  : printf("Unbiased value found in data.\n");
                break;
      case 1  : printf("Wrong data: bias not found\n.");
                break;
      case 2  : printf("EOF found in data.\n");
                break;
      case 3  : printf("Wrong pen selected.\n");
                break;
      case 4  : printf("Invalid number of characters in symbolstring.\n");
                break;
      case 5  : printf("Delta too long. Split not allowed.\n");
                break;
      case 6  : printf("Delta-command expected in controller symbol \n");
                printf("scaling command.\n");
                break;
      case 7  : printf("Incorrect segment alignment in dashline command.\n");
                break;
```

```
      case 8  : printf("Rotation angle too large.\n");
                break;
      case 9  : printf("Wrong operand in mirror-command.\n");
                break;
      case 10 : printf("Wrong operand in window-command.\n");
                break;
      case 11 : printf("Wrong operands in plot symbols-command.\n");
                break;
      case 12 : printf("Count too high in message.\n");
                break;
      case 13 : printf("Wrong channel in plotters select.\n");
                break;
      case 14 : printf("The plot copies-command may only be specified\n");
                printf("at the beginning of a program.\n");
                break;
      case 15 : printf("Too many plot copies specified.\n");
                break;
      case 16 : printf("Wrong parameter in setpen-command.\n");
                break;
      case 17 : printf("Wrong image number in negate-command.\n");
                break;
      case 18 : printf("Wrong rate specified in async-command.\n");
                break;
      case 19 : printf("Wrong pattern specified in pattern/hatch-command.\n");
                break;
      case 20 : printf("Wrong parameter specified in setpat-command.\n");
                break;
      case 21 : printf("Wrong paper command.\n");
                break;
      case 22 : printf("Wrong colorpass number.\n");
                break;
      case 23 : printf("Wrong parameters in XSetpen.\n");
                break;
      case 24 : printf("Wrong pattern number in xsetpat-command.\n");
                break;
      case 25 : printf("Wrong parameters in diskIO.\n");
                break;
      case 26 : printf("Wrong parameters in setlevel-command.\n");
                break;
      case 27 : printf("Delta-command expected in raster fill.\n");
                break;
      case 28 : printf("Wrong parameters in user-defined symbol.\n");
                break;
      case 29 : printf("Wrong parameters in plotting symbols-command.\n");
                printf("from selected symbol set.\n");
                break;
      case 30 : printf("Wrong parameters in font select-command.\n");
                break;
      case 31 : printf("Wrong parameters in chordal tolerance-command.\n");
                break;
      case 32 : printf("Unknown direct pass thru-command.\n");
                break;
      case 33 : printf("Wrong parameters in symbol characteristics-command.\n");
                break;
      case 34 : printf("Delta command in plotter symbol scaling expected.\n");
                break;
      case 35 : printf("Wrong type-parameter in dash bypass-command.\n");
                break;
      case 36 : printf("Pass thru 8-bits direct to plotter not allowed.\n");
                break;
      case 37 : printf("Unknown circle command.\n");
                break;
      case 38 : printf("Delta expected in circle or arc-command.\n");
                break;
```

```
      case 39 : printf("Wrong pixelvalue found in rasterfill-command.\n");
                break;
      case 40 : printf("Bad command found.\n");
                break;
     }
   fclose(fp_out);
   exit(num);
  }


void warning(wr)      /* Produceert een waarschuwing, maar het prog. gaat door */
  int wr;
  {
   printf("Warning: ");
   switch(wr)
     {
      case 1  : printf("unknown escape command detected and ignored.\n");
                break;
      case 2  : printf("no EOM detected in header. EOM has been added.\n");
                printf("but a good plot is not guaranteed.\n");
                break;
      case 3  : printf("response request not found and added.\n");
                break;
      case 4  : printf("EOF found while scanning suffix.\n");
                break;
      case 5  : printf("No start plot after a force plot.\n");
                break;
     }
   }


void message(mr)         /* Produceert berichten ter verduidelijking */
  int mr;
  {
   switch(mr)
     {
      case 1  : printf("Search adress ");
                break;
      case 2  : printf("Radix ");
                break;
      case 3  : printf("Buffer ");
                break;
      case 4  : printf("Requeststring ");
                break;
      case 5  : printf("Good response message ");
                break;
      case 6  : printf("Bad response message ");
                break;
      case 7  : printf("Waiting response message ");
                break;
      case 8  : printf("Maximum waiting time ");
                break;
      case 9  : printf("Turnaround delay ");
                break;
      case 10 : printf("Response suffixlength ");
                break;
      case 11 : printf("Pen number ");
                break;
      case 12 : printf("Scale factor ");
                break;
      case 13 : printf("Suffix string ");
                break;
```

```
    }
   printf("set to default value.\n");
  }



getvalue(c)         /* lees een element uit de plotterfile */
  int *c;
  {
   if ((*c=getc(fp_in))==NULL)
     {
      printf("Error in getvalue.\n");
      exit(1);
     }
  }



putvalue(c)         /* schrijft een waarde naar de uitvoerfile */
  int c;
  {
   if ((putc(c,fp_out))==NULL)
     {
      printf("Error in putvalue.\n");
      exit(1);
     }
  }



void getvalue_biased(c)              /* Tracht een biased waarde op te halen */
  int *c;
  {
   getvalue(c);
   if (*c == EOF) data_error(2);
   if (*c < bias) data_error(0);
  }



void getvalue_noEOF(c)               /* Breekt af bij een EOF */
  int *c;
  {
   getvalue(c);
   if (*c == EOF) data_error(2);
  }



void ungetvalue(c)
  int c;
  {
   ungetc(c,fp_in);
  }



getmessage(a)                /* Leest een string uit de file */
  int a[];
```

```
  {
   int count;                /* Aantal karakters in de string PLUS bias */
   int stringlengte;         /* 2*(count-bias) */
   int j;
   int lowbyte, highbyte;    /* Nibbles van een karakter */

   getvalue(&count);         /* Set request string */
   if ((count-bias<1) ||
      (count-bias>15))
      error(2);              /* verkeerde parameter */
   else {
         stringlengte=2*(count-bias);
         a[0]=count;
         for (j=1;j<=stringlengte;j=j+2)
           {
            getvalue(&highbyte);
            getvalue(&lowbyte);
            a[j]=highbyte;
            a[j+1]=lowbyte;
           }
        }
   }
```

```
putmessage(a)                /* Schrijft een string weg in het Calcomp-format */
  int a[];
  {
   int count;                /* Aantal karakters plus bias */
   int stringlengte;         /* Aantal nibbles: twee per karakter */
   int j;
   int lowbyte,highbyte;     /* Nibbles van een karakter */

   count = a[0];
   stringlengte = 2*(count-bias);
   putvalue(count);
   for (j=1;j<=stringlengte;j=j+2)
     {
      highbyte = a[j];
      lowbyte = a[j+1];
      putvalue(highbyte);
      putvalue(lowbyte);
     }
  }
```

```
convert(s,t)        /* Converteer de nibbles in s naar karakters in t */
  int s[],t[];
  {
   int count;
   int stringlengte;
   int highnibble, lownibble;
   int byte;
   int i,j;

    count = s[0];
    stringlengte = 2*(count-bias);
    t[0] = count-bias;
    for (j=1;j<=stringlengte;j=j+2)
```

```
      {
       highnibble = s[j];
       lownibble  = s[j+1];
       byte = (16 * (highnibble - bias)) + (lownibble - bias);
       i = (j/2) + 1;
       requestcode[i] = byte;
      }
  }



void copynbytes(n)          /* Copieert n bytes van in- naar uitvoer */
  int n;
  {
   int j;
   int c;

   for (j=1; j<=n; j=j+1)
     {
      getvalue_biased(&c);
      putvalue(c);
     }
  }



void openfiles(argc,argv)   /* open te in- en uitvoer files */
  int argc;
  char *argv[];

  {
   char *mode_in, *mode_out;

   mode_in="r";        /* read */
   mode_out="w";       /* write */

   progname = argv[0];                      /* de van het programma */
   if (argc<2) error(3);
   else {
         inputfile = argv[1];
         fp_in=fopen(inputfile,mode_in);    /* open filenaam voor input */
         if (fp_in==NULL) error(4);
        }
   if (argc<3) error(7);
   else {
         outputfile = argv[2];
         fp_out=fopen(outputfile,mode_out); /* open uitvoerfile voor output */
         if (fp_out==NULL) error(8);
        }
  }



void defaultvalues()   /* Initialiseert header parameters */
```

```
  {
   /* -1 betekent: niet gedefinieerd in de header */
   sync1 = -1;
   sync2=(-1);
   OriginalEOM=(-1);
   shigh=(-1);
   smid=(-1);
   slow=(-1);
   radix=(-1);
   pennum=(-1);
   scale=(-1);
   buffer=(-1);
   turnaround=(-1);
   waittime=(-1);
   suffixlength=(-1);
   disable_del=FALSE;
   requeststring[0]=0;          /* geen requeststring */
   goodresponse[0]=0;
   badresponse[0]=0;
   waitresponse[0]=0;
  }
```

```
void do_sync_and_bias() /* haal sync-characters op en de bias */
  {
   int c1, c2, c3;

   getvalue_noEOF(&c1);
   switch (c1)
     {
      case 02 : sync1 = 2;
                break;
      case 22 : getvalue_noEOF(&c2);
                if (c2 == 22)
                  {
                   sync1 = 22;
                   sync2 = 22;
                  }
                else
                  error(0);
                break;
      case 12 : getvalue_noEOF(&c2);
                getvalue_noEOF(&c3);
                if ((c2 == 13) && (c3 == 10))
                  {
                   sync1 = 13;
                   sync2 = 10;
                  }
                else
                  error(0);
                break;
      case 32 : getvalue_noEOF(&c2);
                if (c2 == 2)
                  {
                   sync1 = 32;
```

```
                    sync2 = 2;
                   }
                 else
                   error(0);
                 break;
       default : error(0);
                 break;
      }
    getvalue_noEOF(&bias);
   }



void do_escape()        /* verwerk escape sequences */
   {
    int c;
    int lowbyte, highbyte;
    int character;
    int j;
    int stringlengte;

    getvalue(&c);
    switch(c-bias)
      {
       case 0  :                          /* Disable double buffering */
       case 1  :                          /* Enable double buffering */
       case 10 : buffer = 10 + bias;      /* Set 16 buffers */
                 buflen = 128;
                 break;
       case 11 : buffer = 11 + bias;      /* Set 8 buffers */
                 buflen = 256;
                 break;
       case 12 : buffer = 12 + bias;      /* Set 4 buffers */
                 buflen = 512;
                 break;
       case 13 : buffer = 13 + bias;      /* Set 2 buffers */
                 buflen = 1024;
                 break;
       case 6  : getmessage(requeststring);   /* Set request string */
                 break;
       case 3  : getvalue(&turnaround);       /* Set turnaround delay */
                 break;
       case 4  : getmessage(goodresponse);    /* Good response message */
                 break;
       case 5  : getmessage(badresponse);     /* Bad response message */
                 break;
       case 7  : getvalue(&waittime);         /* Waiting time */
                 break;
       case 8  : getmessage(waitresponse);    /* Waiting response */
                 break;
       case 2  : getvalue(&suffixlength);     /* Suffix length */
                 break;
       case 9  : disable_del=TRUE;            /* Disable <DEL> */
                 break;
       default : warning(1);                  /* Onbekend commando */
                 break;
      }
   }
```

```
void do_datasequence()
  {
   int c;
   int EOMFound;

   EOMFound=FALSE;
   getvalue(&c);
   while ((c!=EOF) && (!EOMFound) && (c!=sync1))
     {
       switch(c-bias)
        {
         case 0   : break;                        /* NO Operation */
         case 1   : getvalue_biased(&shigh);       /* Search adress */
                    getvalue_biased(&smid);
                    getvalue_biased(&slow);
                    break;
         case 7   : getvalue_biased(&radix);       /* Radix minus 1 */
                    radix = radix+1-bias;          /* Radix */
                    break;
         case 8   : do_escape();                   /* Escape sequences */
                    break;
         case 4   : getvalue_biased(&pennum);      /* Select pen */
                    break;
         case 9   : getvalue_biased(&scale);       /* Set scaling */
         default  : if (c<bias)
                      {
                       EOMFound=TRUE;              /* EOM gevonden */
                       OriginalEOM=c;
                      }
                    break;
        }
      getvalue(&c);
     }
   if ((c == EOF) && (!EOMFound))
     error(1);                          /* onverwacht einde van de file */
   if (c == sync1) geensuffix=TRUE;     /* geen suffix gevonden */
   else geensuffix=FALSE;
   lastvalue=c;
  }



void do_header()        /* filter informatie uit de header */
  {
   do_sync_and_bias();
   do_datasequence();
  }



void write_sync()                  /* schrijf de SYNC-character(s) weg */
  {
   putvalue(SYNC);
  }
```

```
void write_bias()
  {
   putvalue(bias);
  }


void write_search()
  {
   int bsearch;

   bsearch=1+bias;     /* de biased search opcode */
   if ((shigh!=bias) || (smid!=bias) || (slow!=(bias+1)))
     {
      message(1);
      shigh=bias;
      smid=bias;
      slow=bias+1;     /* De default waarden PLUS de bias */
     }
   putvalue(bsearch);
   putvalue(shigh);
   putvalue(smid);
   putvalue(slow);
  }



void write_radix()
  {
   int bradix;
   int wradix;

   if (radix==-1)
     {
      /* de radix werd niet gedefinieerd */
      radix = 65;               /* De default radix-waarde */
      message(2);
     }
   bradix = 7+bias;             /* het biased RADIX-commando */
   wradix = radix + bias -1;    /* Radix, zoals die wordt weggeschreven */
   putvalue(bradix);
   putvalue(wradix);
  }




void write_escapesequences()
  {
   int bescape,brequest,bgood,bbad,bwait;
   int bdelay,bsuflen,btime,bdelrec;

   bescape=8+bias;          /* biased escape-commando */
   brequest=6+bias;         /* biased requeststring-commando */
   bgood=bias+4;            /* biased good response */
   bbad=bias+5;             /* biased bad response */
   bwait=bias+8;            /* biased wait response */
   bdelay=bias+3;           /* biased turnaround delay */
   bsuflen=bias+2;          /* biased response suffix length */
   btime=bias+7;            /* biased max waiting time */
   bdelrec=bias+9;          /* biased <DEL> recognition */
```

```
   /* schrijf buffer weg */
   if (buffer==-1)
     {
      /* er werd geen buffer gealloceerd */
      buffer=13+bias;      /* 16 128-byte buffers */
      message(3);
     }
   putvalue(bescape);
   putvalue(buffer);


  /* request string */
  if (requeststring[0]==0)
    {
     requestdefined=FALSE;
     requeststring[0]=bias+1;            /* lengte is 1 karakter */
     requeststring[1]=bias+3;
     requeststring[2]=bias+14;           /* string is 3Ehex */
     message(4);
    }
  else requestdefined=TRUE;
  convert(requeststring,requestcode);    /* Zet string om in code */
  putvalue(bescape);
  putvalue(brequest);
  putmessage(requeststring);


  /* Good response message */
  if (goodresponse[0]==0)
    {
     goodresponse[0]=bias+2;
     goodresponse[1]=bias+3;
     goodresponse[2]=bias;
     goodresponse[3]=bias;
     goodresponse[4]=bias+13;           /* default reponse is 0 CR */
     message(5);
    }
  putvalue(bescape);
  putvalue(bgood);
  putmessage(goodresponse);



  /* Bad response */
  if (badresponse[0]==0)
    {
     badresponse[0]=2+bias;
     badresponse[1]=3+bias;
     badresponse[2]=1+bias;
     badresponse[3]=bias;
     badresponse[4]=13+bias;            /* default response is 1 CR */
     message(6);
    }
  putvalue(bescape);
  putvalue(bbad);
  putmessage(badresponse);



  /* Wait response */
  if (waitresponse[0]==0)
    {
```

```
      waitresponse[0]=2+bias;
      waitresponse[1]=3+bias;
      waitresponse[2]=2+bias;
      waitresponse[3]=bias;
      waitresponse[4]=13+bias;          /* default response is 2 CR */
      message(7);
     }
   putvalue(bescape);
   putvalue(bwait);
   putmessage(waitresponse);



   /* Max waiting time */
   if (waittime==-1)
     {
      waittime=bias;
      message(8);
     }
   putvalue(bescape);
   putvalue(btime);
   putvalue(waittime);



   /* Turnaround delay */
   if (turnaround==-1)
     {
      turnaround=bias;
      message(9);
     }
   putvalue(bescape);
   putvalue(bdelay);
   putvalue(turnaround);



   /* Response suffix length */
   if (suffixlength==-1)
     {
      suffixlength=bias;
      message(10);
     }
   putvalue(bescape);
   putvalue(bsuflen);
   putvalue(suffixlength);



   /* Disable <DEL> recognition */
   if (disable_del)
     {
      putvalue(bescape);
      putvalue(bdelrec);
     }
 }




void write_pen_and_scale()
```

```
  {
   int bpen,bscale;

   bpen=bias+4;              /* biased set pen-command */
   bscale=bias+9;            /* biased set scale-command */
   if (pennum==-1)
     {
      pennum=bias+1;         /* default pennummes */
      message(11);
     }
   putvalue(bpen);
   putvalue(pennum);
   if (scale==-1)
     {
      scale=bias+1;          /* default scalefactor */
      message(12);
     }
   putvalue(bscale);
   putvalue(scale);
  }



void write_EOM()
  {
   if (OriginalEOM==-1)
     warning(2);             /* Geen EOM gevonden: gevaarlijk! */
   putvalue(EOM);
  }



int loadarray(buf)
  int buf[];
  {
   int i;
   int count;

   i = 0;
   count = requestcode[0];
   while ((lastvalue != EOF) && (lastvalue != sync1) && (i != count))
     {
      buf[i] = lastvalue;
      i=i+1;
      if (i != count) getvalue(&lastvalue);
     }
   buf[i] = '\0';
   if (i == count)
     return(TRUE);
   else
     return(FALSE);
  }



int match(buf)                    /* Komt buf overeen met requestcode? */
  int buf[];
  {
   int i;
   int count;
   int matched;
```

```
   matched = TRUE;
   i = 0;
   count = requestcode[0];        /* Aantal tekens in requestcode */
   while ((matched) && (i < count))
     {
      if (buf[i] == requestcode[i+1])
        i=i+1;
      else matched = FALSE;
     }
    return(matched);
  }


void schuifarray(buf)             /* Verschuif het array cyclisch */
  int buf[];
  {
   int count;
   int j;

   count = requestcode[0];
   putvalue(buf[0]);
   for (j=1; j<count; j=j+1)
      buf[j-1] = buf[j];
   buf[count-1] = lastvalue;
  }




void druk_array_af(buf)           /* Stuur buf naar de uitvoerfile */
  int buf[];
  {
   int j;

   j = 0;
   while (buf[j] != '\0')
     {
      putvalue(buf[j]);
      j=j+1;
     }
  }




void copy_suffix()
  {
   int ga_door;
   int gevonden;
   int tempbuf[16];

   if (geensuffix)
     {
      fprintf(fp_out,"Plotting begins...");
      message(13);
     }
   else
     {
      /* EOMFound=TRUE */
      if (requestdefined)
        {
         if (loadarray(tempbuf))
           {
```

```
            ga_door = TRUE;
            gevonden = FALSE;
            while (ga_door)
              {
               if (match(tempbuf))
                 {
                  gevonden = TRUE;
                  ga_door = FALSE;
                 }
               else
                 {
                  getvalue(&lastvalue);
                  if ((lastvalue == EOF) || (lastvalue == sync1))
                    {
                     ga_door = FALSE;
                     druk_array_af(tempbuf);
                    }
                  else schuifarray(tempbuf);
                 }
              }
            }
          else druk_array_af(tempbuf);
          if (!gevonden)
            warning(3);
         }
       else
         {
          while ((lastvalue!=EOF) && (lastvalue!=sync1))
            {
             putvalue(lastvalue);
             getvalue(&lastvalue);
            }
         }
      }
    }



void write_responserequest()
  {
   int lengte, j;

   lengte = requestcode[0];
   for (j=1;j<=lengte;j=j+1)
     {
      putvalue(requestcode[j]);
     }
  }




void create_header()
  {
   write_sync();
   write_bias();
   write_search();
   write_radix();
   write_escapesequences();
   write_pen_and_scale();
   write_EOM();
   copy_suffix();
   write_responserequest();
```

```
  }



int find_sync()
  {
   int ga_door;
   int resultaat;


   ga_door = TRUE;
   while (ga_door)
     {
      if (lastvalue == EOF)                  /* Geen SYNC gevonden */
        {
         ga_door = FALSE;
         resultaat = FALSE;
        }
      else
        {
         if (lastvalue == sync1)             /* Eerste SYNC-character gevonden */
           {
            if (sync2 == -1)                 /* Er is geen tweede SYNC-char */
              {
               ga_door = FALSE;
               resultaat = TRUE;
              }
            else                             /* Tweede SYNC-char verwacht */
              {
               getvalue(&lastvalue);
               if (lastvalue == sync2)       /* Tweede SYNC-char gevonden */
                 {
                  ga_door = FALSE;
                  resultaat = TRUE;
                 }
              }
           }
         else                                /* Niet gevonden: zoek verder */
           getvalue(&lastvalue);
        }
     }
    return(resultaat);
  }



void process_bias()
  {
   getvalue(&lastvalue);
   if (lastvalue != bias)                   /* Eerste teken na de SYNC moet */
     data_error(1);
   else write_bias();
  }



void searchadress()                         /* Search adress */
```

```
  {
   int c;

   getvalue_noEOF(&c);
   if (c >= bias)
     {
      putvalue(1+bias);                    /* Biased search adress-command */
      putvalue(c);
      copynbytes(2);
     }
   else
     if (c == OriginalEOM)
       ungetvalue(c);
     else
       data_error(0);
  }



void penselect()                           /* Pen select: code 4 */
  {
   int c;

   getvalue_noEOF(&c);
   if (c >= bias)
     {
      if (((c-bias) < 1) ||
          ((c-bias) > 16)
         )
        data_error(3);                        /* Verkeerd pennummer */
      else
        {
         putvalue(4+bias);                    /* Biased pen select-command */
         putvalue(c);
        }
     }
   else
     if (c == OriginalEOM)
       ungetvalue(c);
     else
       data_error(0);
  }



void scaling()                             /* Set scale: code 9 */
  {
   int c;

   getvalue_noEOF(&c);
   if (c >= bias)
     {
      putvalue(9+bias);                    /* biased scaling-command */
      putvalue(c);
     }
   else
     if (c == OriginalEOM)
       ungetvalue(c);
     else
       data_error(0);
   }
```

```
void process_x_and_y(code,max,split,nx,ny)
  int code;         /* Opcode van delta-command */
  int max;          /* maximum delta-waarde */
  int split;        /* Evt. opsplitsen in meerdere delta-commands */
  int nx, ny;       /* Aantal x- en y-componenten */
  {
   int x[3];        /* De drie x-parameters */
   int y[3];        /* De drie y-parameters */
   int stepx;       /* De door array x gerepresenteerde waarde */
   int stepy;       /* de door array y gerepresenteerde waarde */
   int deltax;
   int deltay;
   int highx, highy;
   int midx, midy;
   int lowx, lowy;
   int repeat;
   int i,j;
   int c;


   for (j=0; j<3; j=j+1)
     {
      x[j] = 0;
      y[j] = 0;
     }
   for (j=0; j<nx; j=j+1)              /* Lees de x-parameters */
     {
      getvalue_biased(&c);
      x[3-nx+j] = c - bias;
     }
   for (j=0; j<ny; j=j+1)              /* Lees de y-parameters as*/
     {
      getvalue_biased(&c);
      y[3-ny+j] = c - bias;
     }
   stepx = (x[0]*radix*radix) + (x[1]*radix) + x[2];
   stepy = (y[0]*radix*radix) + (y[1]*radix) + y[2];
   if (stepx<stepy)
     repeat = (stepy/max) + 1;
   else
     repeat = (stepx/max) + 1;
   if ((split == FALSE) && (repeat>1))
     data_error(5);
   deltax = stepx/repeat;
   deltay = stepy/repeat;
   x[0] = deltax/(radix*radix);
   x[1] = (deltax - (x[0]*radix*radix))/radix;
   x[2] = deltax  - (x[0]*radix*radix) - (x[1]*radix);
   y[0] = deltay/(radix*radix);
   y[1] = (deltay - (y[0]*radix*radix))/radix;
   y[2] = deltay - (y[0]*radix*radix) - (y[1]*radix);
   for (j=0; j<3; j=j+1)
     {
      x[j] = x[j] + bias;
      y[j] = y[j] + bias;
     }
   for (j=1; j<=repeat;  j=j+1)
     {
      putvalue(code);
      for (i=0; i<nx; i=i+1)
```

```
        putvalue(x[3-nx+i]);      /* Aangepaste x-parameters */
      for (i=0; i<ny; i=i+1)
        putvalue(y[3-ny+i]);      /* Aangepaste y-parameters */
    }
  }



void do_delta(code,max,split)             /* Delta command */
  int code;              /* Delta code */
  int max;               /* Maximum waarde voor delta */
  int split;             /* Een delta eventueel splitsen in meerdere? */
  {
   int nx,ny;            /* Aantal x- en y-parameters voor delta */

   switch (code-bias)
    {
     case 16 :
     case 17 :
     case 18 :
     case 19 : nx = 3;
               ny = 3;
               break;
     case 48 :
     case 49 :
     case 50 :
     case 51 : nx = 3;
               ny = 2;
               break;
     case 52 :
     case 53 :
     case 54 :
     case 55 : nx = 3;
               ny = 1;
               break;
     case 40 :
     case 41 :
     case 42 :
     case 43 : nx = 2;
               ny = 3;
               break;
     case 20 :
     case 21 :
     case 22 :
     case 23 : nx = 2;
               ny = 2;
               break;
     case 60 :
     case 61 :
     case 62 :
     case 63 : nx = 2;
               ny = 1;
               break;
     case 44 :
     case 45 :
     case 46 :
     case 47 : nx = 1;
               ny = 3;
               break;
     case 56 :
     case 57 :
     case 58 :
```

```
    case 59 : nx = 1;
              ny = 2;
              break;
    case 24 :
    case 25 :
    case 26 :
    case 27 : nx = 1;
              ny = 1;
              break;
    case 28 :
    case 31 : nx = 0;
              ny = 3;
              break;
    case 32 :
    case 35 : nx = 0;
              ny = 2;
              break;
    case 36 :
    case 39 : nx = 0;
              ny = 1;
              break;
    case 37 :
    case 38 : nx = 1;
              ny = 0;
              break;
    case 33 :
    case 34 : nx = 2;
              ny = 0;
              break;
    case 29 :
    case 30 : nx = 3;
              ny = 0;
              break;
   }
   process_x_and_y(code,max,split,nx,ny);
  }



void symbolstring()                          /* Symbol string count : code 5 */
  {
   int count;
   int c;
   int i;



   getvalue_noEOF(&c);
   if (c >= bias)
     {
      count = c;
      if (((count-bias) <1) ||
          ((count-bias) >= radix )
         )
        data_error(4);                       /* Verkeerd aantal tekens */
      else
        {
         putvalue(5+bias);                   /* Biased symbol string-command */
         putvalue(count);
         for (i=1; i<=(count-bias); i=i+1)
           {
            getvalue_biased(&c);
            putvalue(c);
```

```
          }
        }
      }
    else
      if (c == OriginalEOM)
        ungetvalue(c);
      else
        data_error(0);
  }



void contsymbolscaling()                    /* Controller symbol scaling */
  {
   int deltacom;
   {
    putvalue(6+bias);                       /* Biased symbol scaling-command*/
    getvalue_biased(&deltacom);
    if (((deltacom - bias)<16) ||
        ((deltacom - bias)>63)
       )
      data_error(6);
    do_delta(deltacom,16383,FALSE);         /* Delta-command. Split is */
                                            /* NIET toegestaan. */
   }
  }



void dashline()                             /* Dashline: code 13 */
  {
   int count;
   int deltacode;
   int j;
   int c;

   getvalue_noEOF(&c);
   if (c >= bias)
     {
      count = c;
      putvalue(13+bias);
      putvalue(count);
      count = count - bias;                 /* Trek de bias eraf */
      if (count > 1)
        {
         for (j=1; j<=(count/2); j=j+1)
           {                                /* Handel de segment-paren af */
            getvalue_biased(&deltacode);
            if (((deltacode-bias)<28) ||    /* zijn het wel paren? */
                ((deltacode-bias)>39)
               )
              do_delta(deltacode,32767,FALSE);
            else data_error(7);
           }
         if (count%2 == 1)
           {                                /* Verwerk laatste segment */
            getvalue_biased(&deltacode);
            switch (deltacode - bias)       /* Er mag geen y aanwezig zijn */
              {
               case 29 :
               case 30 :
```

```
                case 33 :
                case 34 :
                case 37 :
                case 38 : do_delta(deltacode,32767,FALSE);
                          break;
                default : data_error(7);
                          break;
                }
            }
        }
      }
    else
      if (c == OriginalEOM)
        ungetvalue(c);
      else
        data_error(0);
  }


void rotation()                          /* Het rotation command */
  {
   int highangle;                        /* Hoogste 5 bits van hoek */
   int lowangle;                         /* Laagste 4 bits van hoek */
   int angle;                            /* De rotatiehoek */

   getvalue_biased(&highangle);
   getvalue_biased(&lowangle);
   angle = (16 * (highangle - bias)) + (lowangle - bias);
   if (angle >= 360)
     data_error(8);
   else
     {
      putvalue(31+bias);                 /* Biased rotation-command */
      putvalue(highangle);
      putvalue(lowangle);
     }
  }


void mirror()                            /* Het mirror command */
  {
   int c;

   getvalue_biased(&c);
   if ((c - bias)>3)
     data_error(9);
   else
     {
      putvalue(29+bias);                 /* Biased mirror-command */
      putvalue(c);
     }
  }



void window()                            /* Het window-command */
  {
   int inex1, inex2;
   int deltacode;
   int c;

   getvalue_biased(&inex1);              /* Window in- of exclusief? */
```

```
   if ((inex1-bias)>1)
     data_error(10);
   else
     {
      getvalue_biased(&c);
      if (c != bias)
        data_error(10);
      else
        {
         putvalue(30+bias);
         putvalue(inex1);
         putvalue(bias);
         getvalue(&deltacode);
         if (((deltacode - bias)<16) ||
             ((deltacode - bias)>63)
            )
           data_error(10);              /* Het is geen delta-command */
         else
           {
            do_delta(deltacode,32767,FALSE);
            getvalue_biased(&c);
            if (c != (14 + bias))
              data_error(10);
            else
              {
               getvalue_biased(&c);
               if (c != (30+bias))
                 data_error(10);
               else
                 {
                  getvalue(&inex2);
                  if (inex1 != inex2)
                    data_error(10);
                  else
                    {
                     getvalue(&c);
                     if (c != (1+bias))
                       data_error(10);
                     else
                       {
                        putvalue(14+bias);
                        putvalue(30+bias);
                        putvalue(inex2);
                        putvalue(1+bias);
                       }
                    }
                 }
              }
           }
        }
     }
   }



void plotsymbols()                      /* Het plot symbols command */
  {
   int highnum,lownum;

   getvalue_biased(&highnum);
   getvalue_biased(&lownum);
   if (((highnum-bias)>=32) ||
       ((lownum-bias)>=16)
```

```
      }
     data_error(11);
   else
     {
      putvalue(15+bias);                    /* Biased plot symbols-command */
      putvalue(highnum);
      putvalue(lownum);
     }
  }



void operatormessage(code)                  /* Operator message command */
  int code;
  {
   int count;
   int c;
   int j;

   putvalue(code+bias);                     /* Biased opcode */
   getvalue_biased(&count);
   putvalue(count);
   if ((count-bias)>64)
     data_error(12);
   else
     {
      for  (j=1; j<=(count-bias); j=j+1)
        {
         getvalue_biased(&c);
         putvalue(c);
        }
     }
  }



void plotterselect()                        /* Plotter select */
  {
   int plotnum;

   getvalue_biased(&plotnum);
   if ((plotnum-bias)>7)
     data_error(13);
   else
     {
      putvalue(26+bias);                    /* Biased plotter select-command */
      putvalue(plotnum);
     }
  }



void plotcopies()                           /* Plot copies command */
  {
   int count;

   getvalue_biased(&count);
   if (((count-bias) == 0) ||
       ((count-bias) >63)
      )
     data_error(15);
   else
```

```
    {
     putvalue(27+bias);                    /* Biased plot copies command */
     putvalue(count);
    }
  }




void setpen()                              /* Het setpen-command */
  {
   int weight;
   int width;
   int pennum;

   getvalue_biased(&weight);
   getvalue_biased(&width);
   getvalue_biased(&pennum);
   if (((weight-bias) == 0)  ||
       ((weight-bias) > 16)  ||
       ((width-bias) == 0)   ||
       ((width-bias) > 16)   ||
       ((pennum-bias) == 0)  ||
       ((pennum-bias) > 16)
      )
     data_error(16);
   else
     {
      putvalue(18+bias);                   /* Biased setpen-command */
      putvalue(weight);
      putvalue(width);
      putvalue(pennum);
     }
  }




void negate()                              /* Het negate command */
  {
   int image;

   getvalue_biased(&image);
   if ((image-bias) >1)
     data_error(17);
   else
     {
      putvalue(17+bias);                   /* Biased negate-command */
      putvalue(image);
     }
  }




void async()                               /* het async-command */
  {
   int highrate;
   int lowrate;

   getvalue_biased(&highrate);
```

```
   getvalue_biased(&lowrate);
   if (((highrate-bias)>100) ||
       ((lowrate-bias)>100)
      )
     data_error(18);
   else
     {
      putvalue(20+bias);
      putvalue(highrate);
      putvalue(lowrate);
     }
  }



void pattern()                          /* Het pattern/hatch command */
  {
   int pat;
   int highvert;
   int lowvert;

   getvalue_biased(&pat);
   getvalue_biased(&highvert);
   getvalue_biased(&lowvert);
   if ((pat-bias)>47)
     data_error(19);
   else
     {
      putvalue(21+bias);                /* Biased pattern-command */
      putvalue(pat);
      putvalue(highvert);
      putvalue(lowvert);
     }
   }



void setpat()                           /* Het setpat  command */
  {
   int pat;
   int line;
   int byte;
   int i;

   putvalue(22+bias);                   /* Biased setpat-command */
   getvalue_biased(&pat);
   if (((pat-bias)<16) ||
       ((pat-bias)>47)
      )
     data_error(20);
   else
     {
      putvalue(pat);                    /* Het patroon */
      getvalue(&line);
      if ((line-bias)>31)
        data_error(20);
      else
        {
         putvalue(line);                /* Welke rij? */
         copynbytes(4);                 /* Copieer de volgende 4 bytes */
        }
```

```
    }
  }


void paper()                                /* De paper commands */
  {
   int c;

   getvalue_biased(&c);
   if ((c-bias)>1)
     data_error(21);
   else
     {
      putvalue(19+bias);                    /* Biased paper-command */
      putvalue(c);
     }
  }



void forceplot()                            /* Het force plot command */
  {
  /* Een force plot MOET gevolgd worden door een start plot. */
  int c;

  putvalue(25+bias);                        /* Biased force plot-command */
  putvalue(14+bias);                        /* Biased extended-command */
  putvalue(24+bias);                        /* Biased start plot-command */
  getvalue(&c);
  if ((c-bias) != 14)
    {
     ungetvalue(c);
     warning(5);
    }
  else
    {
     getvalue(&c);
     if ((c-bias) != 24)
       {
        warning(5);
        ungetvalue(c);
        do_extended();
       }
    }
  }



void newpen()                               /* Het newpen-command */
  {
   putvalue(32+bias);                       /* Biased newpen-command */
   copynbytes(3);
  }



void colorsequence()                        /* Color sequence-command */
  {
   int colorpass;
```

```
   getvalue_biased(&colorpass);
   if (((colorpass-bias) == 0) ||
       ((colorpass-bias) > 10)
      )
     data_error(22);
   else
     {
      putvalue(36+bias);                    /* Biased color sequence-command */
      putvalue(colorpass);
     }
  }


void patternfill()                          /* Het pattern fill-command */
  {
   putvalue(33+bias);                      /* Biased pattern fill-command */
   copynbytes(5);
  }



void areafill()                             /* Het area fill-command */
  {
   putvalue(44+bias);                      /* Biased area fill-command */
   copynbytes(3);
  }


void colormodify()                          /* Het color modify-command */
  {
   putvalue(35+bias);                      /* Biased color modify-command */
   copynbytes(11);
  }



void xsetpen()                              /* Het XSetpen-command */
  {
   int j;
   int c;

   putvalue(40+bias);                       /* Biased xsetpen-command */
   getvalue_biased(&c);
   if ((c-bias) != 0)
     error(23);
   else
     {
      copynbytes(3);
      for (j=1; j<=13; j=j+1)
        {
         getvalue_biased(&c);
         if (((c-bias) == 0) ||
             ((c-bias) > 16)
            )
           error(23);
         else
           putvalue(c);
        }
     }
  }
```

```
void xsetpat()                          /* XSetpattern-command */
  {
   int midpat;
   int lowpat;
   int patnum;
   int c;

   getvalue_biased(&midpat);
   getvalue_biased(&lowpat);
   patnum = ((midpat - bias) * radix) + (lowpat - bias) + 15;
   if ((patnum <16) ||
       (patnum >239)
      )
     data_error(24);
   else
     {
      putvalue(34+bias);                /* Biased XSetpattern-command */
      putvalue(midpat);
      putvalue(lowpat);
      getvalue_biased(&c);
      if ((c-bias) >31)
        error(24);
      else
        {
         putvalue(c);
         copynbytes(8);
        }
     }
   }



void diskio()                           /* Het diskIO-command */
  {
   int usr;
   int save_or_restore;
   int what;

   getvalue_biased(&usr);
   if ((usr-bias) >9)
     data_error(25);
   else
     {
      getvalue_biased(&save_or_restore);
      if (((save_or_restore-bias) != 1) &&
          ((save_or_restore-bias) != 2)
         )
        data_error(25);
      else
        {
         getvalue_biased(&what);
         if (((what-bias) == 0) ||
             ((what-bias) > 7)
            )
           data_error(25);
         else
           {
            putvalue(37+bias);          /* Biased diskio-command */
            putvalue(usr);
            putvalue(save_or_restore);
            putvalue(what);
```

```
          }
        }
      }
   }


void setlevel()                          /* Setlevel-command */
  {
   int c;

   putvalue(41+bias);                    /* Biased setlevel-command */
   copynbytes(2);
   getvalue_biased(&c);
   if ((c-bias) > 2)
     data_error(26);
   else putvalue(c);
  }


void newlevel()                          /* Newlevel-command */
  {
   putvalue(38+bias);                    /* Biased newlevel-command */
   copynbytes(2);
  }



void rasterfill()                        /* Rasterfill */
  {
   int j;
   int c;
   int pix;

   putvalue(42+bias);                    /*  Biased rasterfill-command */
   for (j=1; j<=2; j=j+1)
     {
      getvalue_biased(&c);
      if (((c-bias) < 16) ||
          ((c-bias) > 63)
         )
        data_error(27);
      else do_delta(c,32767,FALSE);
     }
    getvalue_biased(&pix);
    if (((pix-bias) == 0) ||
        ((pix-bias) > 3)
       )
      data_error(39);
    else
      {
       putvalue(pix);
       getvalue_biased(&c);
       if (((c-bias) < 16) ||
           ((c-bias) > 63)
          )
         data_error(27);
       else do_delta(c,32767,FALSE);
      }
   }
```

```
void pixel()                                /* Het pixel-command */
  {
   int count;

   putvalue(43+bias);                       /* Biased pixel-command */
   getvalue_biased(&count);                  /* Aantal volgende bytes */
   putvalue(count);
   copynbytes(count-bias);
  }




void userdefinedsymbol()
  {
   int cnum;
   int pnum;
   int count;

   getvalue_biased(&cnum);
   getvalue_biased(&pnum);
   if (((cnum-bias) > 5) ||
       ((pnum-bias) >15)
      )
     data_error(28);
   else
     {
      getvalue_biased(&count);
      putvalue(13+bias);                     /* Biased user symbol-command */
      putvalue(cnum);
      putvalue(pnum);
      putvalue(count);
      copynbytes(2*(count - bias));
     }
  }



void xsymbol(sym)                           /* Symbol commnads */
  int sym;
  {
   int c;

   putvalue(sym+bias);
   getvalue_biased(&c);
   if ((c-bias) != 14)                      /* het is geen extended-command  */
     ungetvalue(c);
   else
     {
      putvalue(c);                         /* Baised extended-command */
      getvalue_biased(&c);
      if ((c-bias) == 13)
         userdefinedsymbol();
      else
       if ((c-bias) == 14)
          putvalue(c);
       else
         {
          ungetvalue(c);
          do_extended();
```

```
        }
      }
    }


void plotsymbol(cnum)
  int cnum;
  {
   int pnum;

   getvalue_biased(&pnum);
   if (((cnum-bias) > 5) ||
       ((pnum-bias) >15)
      )
     data_error(29);
   else
     {
      putvalue(cnum);
      putvalue(pnum);
     }
  }


void do_extended()                          /* Extended codes */
  {
   int c;


   getvalue_noEOF(&c);
   if (c >= bias)
     {
      putvalue(14+bias);                    /* Biased extended-command */
      switch (c-bias)
        {
         case 31 : rotation();              /* Rotation: code 1Fhex */
                   break;
         case 29 : mirror();                /* Mirror: code 1Dhex */
                   break;
         case 30 : window();                /* Window: code 1EHex */
                   break;
         case  8 :                          /* Symbol set 1: code 08Hex */
         case  9 :                          /* Symbol set 2: code 09Hex */
         case 10 :                          /* Symbol set 3: code 0AHex */
                   putvalue(c);
                   break;
         case 15 : plotsymbols();           /* Plot symbols: code 0FHex */
                   break;
         case  6 : operatormessage(6);      /* Message: code 06Hex */
                   break;
         case  7 : operatormessage(7);      /* Message: code 07Hex */
                   break;
         case 26 : plotterselect();         /* Plotter select: code 1Ahex */
                   break;
         case 27 : data_error(14);          /* Plot copies mag alleen aan */
                   break;                   /* het begin staan */
         case 18 : setpen();                /* Set pen: code 12Hex */
                   break;
         case 17 : negate();                /* Negate: code 11Hex */
                   break;
         case 20 : async();                 /* Async: code 14Hex */
                   break;
         case 21 : pattern();               /* Pattern: code 15Hex */
```

```
                break;
        case 22 : setpat();                 /* Setpat: code 16Hex */
                break;
        case 19 : paper();                  /* Paper-command: code 13Hex */
                break;
        case 24 : putvalue(c);              /* Start plot: code 18Hex */
                break;
        case 25 : forceplot();              /* Force plot: code 19Hex */
                break;
        case 32 : newpen();                 /* New pen: code 20Hex */
                break;
        case 36 : colorsequence();          /* Color sequence: code 24Hex */
                break;
        case 33 : patternfill();            /* Pattern fill: code 21Hex */
                break;
        case 44 : areafill();               /* Area fill: code 2CHex */
                break;
        case 35 : colormodify();            /* Color modify: code 23Hex */
                break;
        case 40 : xsetpen();                /* XSetpen: code 28Hex */
                break;
        case 34 : xsetpat();                /* XSetpattern: code 22Hex */
                break;
        case 37 : diskio();                 /* DiskIO: code 25Hex */
                break;
        case 41 : setlevel();               /* Setlevel: code 29Hex */
                break;
        case 38 : newlevel();               /* Newlevel: code 26Hex */
                break;
        case 42 : rasterfill();             /* Rasterfill: code 2AHex */
                break;
        case 43 : pixel();                  /* Pixel: code 2BHex */
                break;
        case 39 : putvalue(c);              /* Plot status: code 27Hex */
                break;
        case 11 : xsymbol(11);              /* Symbol set 3: code 0BHex */
                break;
        case 12 : xsymbol(12);              /* Symbol set 4: code 0CHex */
                break;
        default : plotsymbol(c);            /* Plot symbol: code 0EHex */
                break;
      }
    }
   else
     if (c == OriginalEOM)
       ungetvalue(c);
     else
       data_error(0);
  }



void plotterperformance()                   /* Het plotter performance */
  /* Het plotter performance-command verschilt per controller.
     Voor de elektrostaat is dit command NIET van toepassing. */

  {
   int c;
   int j;

   putvalue(4+bias);                        /* Vervang 'm door een NOP */
   getvalue_biased(&c);
```

```
   if ((c-bias) == 11)                    /* PCI - controller? */
      for (j=1; j<=5; j=j+1)
        getvalue_biased(&c);               /* Sla de rest over */
   else                                   /* Een 907-controller */
      for (j=1; j<=3; j=j+1)
        getvalue_biased(&c);
  }



void fontselect()
  {
   int c1,c2;

   putvalue(9+bias);                      /* Biased font  select-command */
   getvalue_biased(&c1);
   getvalue_biased(&c2);
   if (((c1-bias) != 11) ||
       ((c2-bias) != 7)
      )
     data_error(30);
   else
     {
      putvalue(c1);
      putvalue(c2);
      getvalue_biased(&c1);
      if ((c1-bias) == 14)
        {
         getvalue_biased(&c2);
         if ((c2-bias) != 8)
           data_error(30);
         else
           {
            putvalue(c1);
            putvalue(c2);
           }
        }
       else
         if (((c1-bias) == 0) ||
             ((c1-bias) > 10)
            )
           data_error(30);
         else
           putvalue(c1);
     }
  }



void chordaltolerance()                   /* Het chordal tolerance-command */
  {
   int c1, c2;

   putvalue(8+bias);                      /* Biased chordal tolerance-command */
   getvalue_biased(&c1);
   if ((c1-bias) != 11)
     data_error(31);
   else
```

```
    {
     putvalue(c1);
     getvalue_biased(&c1);
     getvalue_biased(&c2);
     if (((c1-bias) >9) ||
         ((c2-bias) >9)
        )
       data_error(31);
     else
       {
        putvalue(c1);
        putvalue(c2);
       }
    }
  }



void double_zero()                          /* De 0B 00 XX -commands */
  {
   int c;

   putvalue(bias);                          /* Biased 00-command */
   getvalue_biased(&c);
   switch (c-bias)
     {
      case  5 : plotterperformance();       /* Plotter performance */
                break;
      case  9 : fontselect();               /* Font select */
                break;
      case  8 : chordaltolerance();         /* Chordal tolerance */
                break;
      case  3 :                             /* Manual-command */
      case  4 :                             /* No-Operation */
      case  6 : putvalue(c);                /* New plot */
                break;
      default : data_error(32);             /* Zeker geen command */
                break;
     }
  }



void characteristics()                      /* Symbol characteristics */
  {
   int c;
   int j;
   int ndelta;                              /* Aantal delta's om tekens */
                                            /* te definieren */
   putvalue(5+bias);                        /* Biased command */
   getvalue_biased(&c);
   if ((c-bias) == 15)
     ndelta = 4;
   else
     if ((c-bias) == 14)
       ndelta = 3;
     else
       data_error(33);
   putvalue(c);                             /* Wel of geen slant ? */
   for (j=1; j<=ndelta; j=j+1)
```

```
    {
     getvalue_biased(&c);
     if (((c-bias) < 16) ||
         ((c-bias) > 63)
        )
       data_error(33);                    /* Geen delta-command */
     else
       do_delta(c,32767,FALSE);
    }
  }



void symbolscaling()                      /* Symbol scaling-command */
  {
   int c;

   putvalue(7+bias);                      /* Biased symbol scaling-command */
   getvalue_biased(&c);
   if ((c-bias) == 6)
     getvalue_biased(&c);
   if (((c-bias) < 16) ||
       ((c-bias) > 63)
      )
     data_error(34);
   else
     do_delta(c,32767,FALSE);
  }



void dashbypass()                         /* Dash bypass-command */
  {
   int  c;

   putvalue(8+bias);                      /* Biased dash bypass-command */
   getvalue_biased(&c);
   if (((c-bias) == 0) ||
       ((c-bias) == 1) ||
       ((c-bias) == 3) ||
       ((c-bias) == 5) ||
       ((c-bias) == 7)
      )
     putvalue(c);
   else
     data_error(35);
  }



void do_direct()                          /* Direct thru plotter-commands */
  {
   int c;

   getvalue_noEOF(&c);
   if (c >= bias)
     {
      putvalue(11+bias);                  /* Biased Direct thru-command */
      switch (c-bias)
        {
```

```
        case  0 : double_zero();        /* De 0B 00 XX -commands */
                  break;
        case  5 : characteristics();    /* Symbol characteristics */
                  break;
        case  7 : symbolscaling();      /* Symbol scaling */
                  break;
        case  8 : dashbypass();         /* Dash bypass */
                  break;
        default : data_error(36);       /* Is het misschien een pass thru? */
                  break;
      }
    }
  else
    if (c == OriginalEOM)
      ungetvalue(c);
    else
      data_error(0);
  }



void figure(t,n)                        /* Verwerk een circle of arc */
  int t;                                /* code van circle of arc */
  int n;                                /* Aantal bijbehorende deltas */
  {
   int j;
   int c;

   putvalue(t);
   for (j=1; j<=n; j=j+1)
     {
      getvalue_biased(&c);
      if (((c-bias) < 16)  ||
          ((c-bias) > 63)
         )
        data_error(38);
      else
        do_delta(c,32767,FALSE);
     }
  }



void do_circle_and_arc()                /* De circle- en arc-commands */
  {
   int c;


   getvalue_noEOF(&c);
   if (c >= bias)
     {
      putvalue(12+bias);                /* Biased circle-command */
      switch (c-bias)
       {
        case  0 :
        case  1 :
        case  2 :
        case  3 : figure(c,2);          /* 16-bit arc: 2 deltas */
                  break;
        case  4 :
        case  5 :
        case  6 :
```

```
        case  7 : figure(c,1);          /* 16-bit circle: 1 delta */
                  break;
        case  8 :
        case  9 :
        case 10 :
        case 11 : figure(c,4);          /* 24-bit arc: 4 deltas */
                  break;
        case 12 :
        case 13 :
        case 14 :
        case 15 : figure(c,2);          /* 24-bit circle: 2 deltas */
                  break;
        default : data_error(37);
                  break;
       }
     }
   else
     if (c == OriginalEOM)
       ungetvalue(c);
     else
       data_error(0);
  }



void process_datasequence()             /* Verwerk de plotdata... */
  {
   int c;
   int d;

   inbuf = 0;
   getvalue(&c);
   if ((c-bias) == 14)
     {
      inbuf = MAXOPLEN;
      getvalue_noEOF(&c);
      if ((c-bias) == 27)               /* Is het misschien plot copies */
        {
         putvalue(14+bias);
         plotcopies();
        }
      else
        {
         if (c == OriginalEOM)
           ungetvalue(c);
         else
           if (c < bias)
             data_error(0);
           else
             {
              ungetvalue(c);                  /* Vals alarm */
              do_extended();
             }
        }
      getvalue(&c);
     }
   while ((c != EOF) && (c != OriginalEOM))
     {
      if ((inbuf + MAXOPLEN) > (buflen - 2))
        {
```

```
        /* Begin aan een nieuwe datamessage */
        write_EOM();
        write_responserequest();
        write_sync();
        write_bias();
        inbuf = 0;
       }
     /* Parse de codes */
     switch (c-bias)
       {
        case 0  :                        /* No-Operation */
        case 2  :                        /* Pen down */
        case 3  :                        /* Pen up */
        case 10 :                        /* pause */
        case 15 : putvalue(c);           /* End-of-plot */
                  break;
        case 4  : penselect();           /* Pen select */
                  break;
        case 1  : searchadress();        /* Search adress */
                  break;
        case 9  : scaling();             /* Set scale */
                  break;
        case 5  : symbolstring();        /* Symbol string count */
                  break;
        case 6  : contsymbolscaling();   /* Controller symbol scaling */
                  break;
        case 13 : dashline();            /* Dashline */
                  break;
        case 14 : do_extended();         /* Extended commands */
                  break;
        case 11 : do_direct();           /* Pass thru direct to plotter */
                  break;
        case 12 : do_circle_and_arc();   /* Circle en Arc's */
                  break;
        default : getvalue_noEOF(&d);
                  if (d >= bias)
                    {
                     ungetvalue(d);
                     if (((c-bias)< 16) ||
                         ((c-bias)> 63)
                        )
                       data_error(40);
                     else do_delta(c,32767,TRUE); /* Delta-command,split mag*/
                    }
                  else
                    if (d == OriginalEOM)
                      ungetvalue(d);
                    else
                      data_error(0);
                  break;
       }
     inbuf = inbuf + MAXOPLEN;
     getvalue(&c);
    }
  if (c == OriginalEOM)
    getvalue(&c);                                /* Schuif een plaats op */
  lastvalue=c;
 }
```

```
void process_data()
  {
   int er_is_een_message;

   if (find_sync() == TRUE)          /* Dit is de eerste plotmessage */
     {
      er_is_een_message = TRUE;      /* Er is inderdaad een message */
      write_sync();
      write_bias();
      process_datasequence();
     }
   else
     er_is_een_message=FALSE;
   while (find_sync() == TRUE)       /* Zolang er nog data is... */
     {
      write_EOM();
      write_responserequest();
      write_sync();
      write_bias();
      process_datasequence();
     }
   if (er_is_een_message == TRUE)
     {
      putvalue(15+bias);             /* Sluit af met een end-of-plot */
      write_EOM();
      write_responserequest();
     }
  }



main(argc,argv)
  int argc;
  char *argv[];
  {
   openfiles(argc,argv);     /* Open de in- en uitvoer files */
   defaultvalues();          /* Initialiseer */
   do_header();              /* Verwerk data uit header */
   create_header();          /* Maak nieuwe header uit deze data */
   process_data();           /* tijdelijke functie */
   printf("Parsing succesfully completed.\n");
   fclose(fp_out);
  }
```