CHD

Title:

DUET

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 21-V032
Edition: September 1977
Author: Edith Rosenberg

Keywords: RC4000, RC8000, SYSTEM80, database, SODA, DBMS, PRIMULA, SYSDOK, programming language, interpreter, structured programming.

Abstract: This manual describes the DUET system, which comprises a programming language, a compiler, and an interpreter for execution of the compiled program. The programming language DUET is primarily intended for administrative applications especially transaction processing. It covers DBMS operations as well as the reading of input and the printing of output.

English edition: 215 pages

Printed by A/S Regnecentralen, Copenhagen

Table of Contents Page

1. Introduction 6

2. System Outline 8

3. Duet language 11

3.1 Duet Program Structure 12

3.1.1 Duet Head 17
3.1.2 Duet Blocks 18
3.1.3 Duet Instructions 24
3.1.4 Duet Operands 26

3.2 Duet Operations 32

3.2.1 Execute 33
3.2.2 Modify 39
3.2.3 Compute 49
3.2.4 Assign - of 51
3.2.5 If - then 54
3.2.6 Case - of 55
3.2.7 Action - of 56
3.2.8 For - do 58
3.2.9 While - do 59
3.2.10 Getline 60
3.2.11 Read 62
3.2.12 Print 70
3.2.13 DB-operations 80
3.2.14 Select 85
3.2.15 Exit 90
3.2.16 Algol 91

3.3 Maximum limits in duetoperations

4. The Duet Compiler 93

4.1 Program Text and Listing 94
4.2 Activation of the Compiler 97
4.3 Resource Demands 109
4.4 Duet Log 112
4.5 Error messages 114

Table of Contents

		Page
5.	The Duet System in a Control Program	133
5.1	Duet Texts & Algol Block Structure	134
5.2	Initialization & Termination	138
5.2.1	Init_duet1	139
5.2.2	Init_duet2	141
5.2.3	Init_duet maskine	143
5.2.4	Close_duet	146
5.3	Error Procedures	147
5.3.1	Duet Data Errors	148
5.3.2	Duet Programming Errors	152
5.3.3	Duet System Errors	161
5.4	Algol Special Actions	167
5.5	Reserved Algol Names	170
5.5.1	Application - Known Algol Names	171
5.5.2	Inaccessible Algol Names	180
Appendix A.	Syntax Description of the Duet Language	185
Index		198

References

1. RCSL 21-V031: Database80

2. RCSL 21-V019: SODA

3. RCSL 21-V018: DES80-SODA-LD
(Danish edition)

4. RCSL 21-D005: Connected Files System

5. RCSL 21-V024: DES80 - Consultant Guide
(Danish edition)

6. RCSL 21-T006: Teledata

7. RCSL 28-D017: Informal

1. Introduction

Duet System

The Duet system is a general datamatic tool for implementing programs on RC4000/8000, e.g. for SYSTEM80.

The Duet System Comprises

Duetabler

- a compiler, DUETABLER, which compiles programs written in the DUET language, and
- an interpreter, DUET INTERPRETER, which - being copied into the user's control program - executes the Duet program

Duet Language

The Duet language involves special facilities for read in control of data, which makes it considerably easier to write read in checking programs in Duet than in Algol.

Printing

Furthermore, printing of results is especially easy to implement in the Duet language, where the printing system is based on the Primula procedures (which are also applied in GENIUS).

SODA-dbms

Access to the database records is achieved by means of the Soda-dbms (ref. 2), and therefore the Duet language is closely connected with the ld-description

Soda-ld

of the Soda System, Soda-ld.

The Duet language has been described in detail in section 3, and the compiler in section 4.

Duet Interpreter Control Program

The Duet interpreter is the interpreter which executes the Duet program. It is unable to work alone, and has to be copied into the user's control program which takes care of the starting and the termination of the run.

1. Introduction

SYSTEM80

At present two such completed control programs are available in SYSTEM80:

- the 'Telescop' of the data entry system (DES80) and
- the 'Teleop' of the Teledata System.

DES80 (ref. 3 and 5) only needs coding of pure Duet programs whereas for Teledata (ref. 6) user adaptions programmed in Duet can be produced and incorporated into the existing Skeleton Duet program. At this point the Duet programs must obey the rules determined by the control program and the Skeleton program.

Independent Systems

It is, however, possible to make perfectly new application systems by means of the Duet System with an independent Duet program and a control program. Section 5 concerns directions for the establishment of a control program.

2. System Outline

This section gives an outline of the Duet system and its connection with other tool systems:

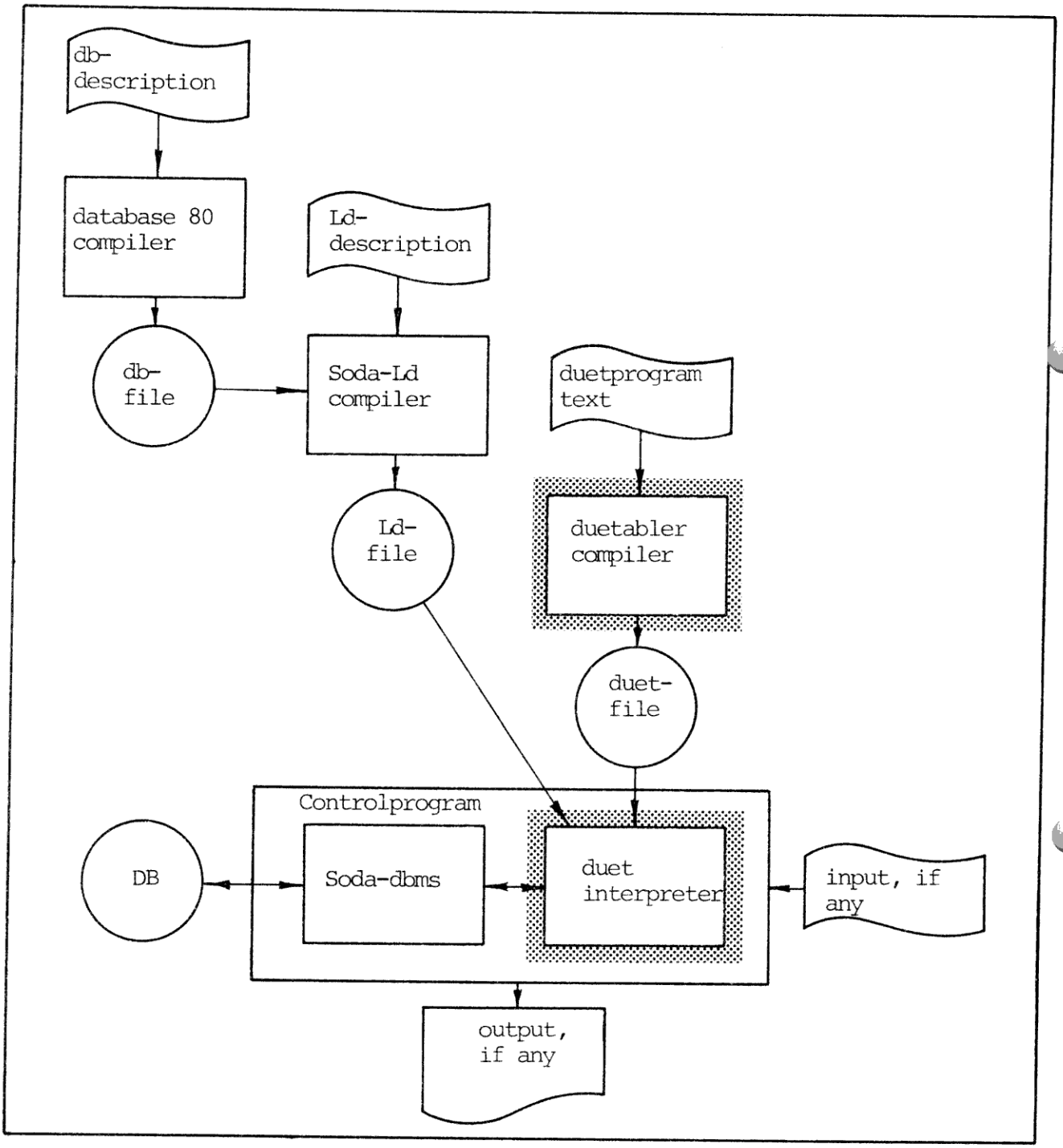


Figure 2.1: Outline of the Duet system's relations to its surroundings.

2. System outline

Figure 2.1 shows the Duet system with its necessary surroundings. A complete application system comprises a control program with the respective ld-file and Duet file.

Database80 db-description

The first step in establishing such an application system is making a data description of the actual database by means of 'database80'. (ref. 1).

Soda-ld: ld-description

Then the relevant subset of the data base must be described in a local data description, which is compiled by the 'soda-ld' into an ld-file (ref. 2).

Duetabler: Duet program

The Duet compiler 'Duetabler' reads a Duet program together with the respective ld-file. The Duet program is checked for syntactical and semantical errors and in case there are no errors a binary Duet program, a Duet file, is generated.

Duet interpreter
Soda-dbms

This translated Duet program can be executed by a control program, which comprises the interpreter: 'Duet interpreter' and the Soda-dbms. It requires access to the ld-file as well as to the Duet file during the run.

Duetblocks

Duetarray

The Duet program consists of smaller units: Duet blocks. These are read, when needed by the Duet interpreter into a virtual store: Duet array, the size of which is decided by the control program. A Duet block is referred to from another Duet block by indicating the block number and an entrypoint, which defines the wanted part of the Duet block.

There are several advantages by this division of the Duet program into blocks:

2. System Outline

- It becomes very easy to implement user adaptions. The adaption point of the Duet skeleton program is merely a reference to some entrypoint in a particular adaption block where the user adaption is programmed.

- The control program may run in a smaller process as it is not necessary to have space in the core for the entire Duet program at the same time.

- The Duetabler compiler is able to compile Duet blocks one by one and connect with an existing Duet file. This facility reduces the compile time when changes are wanted in the Duet program or new user adaptions are to be inserted.

Variable

The Duet program can access all the variables declared in the variable section of the ld-description and they may be referred to either by their name or number.

d-array

The variables exist in a mutual array: the d-array, in which the variable addresses have been allocated by the Soda-ld compiler. Since the Duet program uses these addresses, in every variable reference, it is necessary to re-compile the entire Duet program after any changes in the variable section of the ld-description.

Duet Instruction

Execute list

Each Duet block in a Duet program consists of a number of named Duet instructions. The principle regarding how to execute these instructions by means of so-called execute lists, has been explained in detail in section 3.1.

3. The Duet Language

This section describes the syntax and semantics of the Duet language. The description is based on examples, whereas a formal description of syntax is to be found in appendix A.

3.1 Duet Program Structure

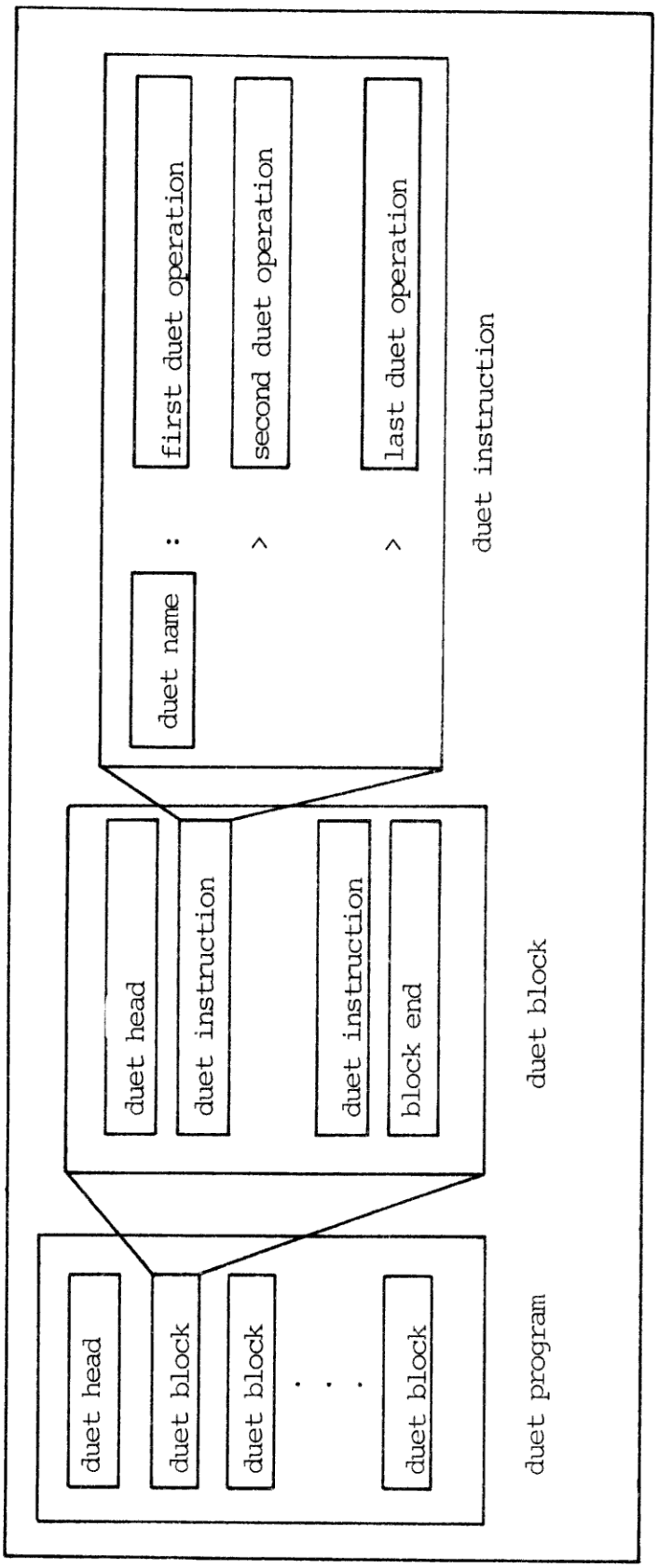


Figure 3.1: Schematic outline of the Duet program structure.

3.1 Duet Program Structure

Figure 3.1 shows the schematic structure of a Duet program as consisting of a Duet head and one or more Duet blocks.

Duet Instruction

Each Duet block consists of a block head, a number of Duet instructions and a block end.

Duet operation
Duet Name

Finally a Duet instruction consists of a Duet name and one or more Duet operations. The Duet name which consists of the letter 'd' followed by an integer, identifies the instruction within the current block. The name is used when the duet instruction is to be executed, analagous to the way in which a procedure identifier is used when a procedure is called from an Algol program

The sequence of the Duet instructions inside a Duet block is arbitrary, analogous to the order of procedure declarations in an Algol block. However, when a Duet instruction is activated, the operations are always executed in the same order as they appear in the instruction, just like the statements in an algol procedure body.

Execute List

The Duet instructions are executed in a sequence, which is defined dynamically during the run of the program. This is done by the execution of a special operation, called an execute list, which contains a list of Duet names (local for the current block). The Duet instructions, mentioned in the list, will be executed in the same sequence as they appear.

3.1 Duet Program Structure

```
d315: execute  d12                          ; execute list
               d240
               d316
               s
   > modify    v_name := 'screws'           ; variable assign
               number :+ 1
               s
   > compute   v33(index) := v45 * (v12 + v16)
               s
   > print     <1 1 t6>   : 'balance'       ; print
               <10 n5.2> : balance
               s
   > if        balance>1000.00 then d318    ; conditional
                                            ; operation
```

Figure 3.2: example of Duet instruction

Figure 3.2 shows an example of a Duet instruction consisting of five operations.

Duet Operator
Duet Operand

Every Duet operation is headed by a Duet operator which designates the following format. After this, follows one or more Duet operands seperated by appropriate delimiters keywords, or numerical operators.

Some Duet operators demand a fixed format with a specific set of Duet operands, whereas other operators can contain a variable number of operands.

In the latter case the operands are organized in a list, the elements of which are normally separamted by line feed. Such a list is always terminated with a separate line with the terminator 's' - the so-called Duet Stop.

Duet Stop

3.1 Duet Program Structure

Comment

All lines in a Duet program may be terminated by a semicolon and a comment text which ends at the line feed. In the printing from the Duetabler compiler, such comments will be edited in a coloumn (see section 4.3). Empty lines may be inserted (perhaps with a comment) anywhere in the program.

Execution of Duet Program

The execution of the instructions within one block as well as within the entire Duet program is performed in a number of dynamically defined levels. Whenever an execution of a Duet instruction is ordered from an execute list, the control is transferred to a new level - in practice a pointer is stacked at the current point in the execute list, which makes it possible later, when the referred instruction is executed, to continue in the list.

When the last instruction of an execute list has been executed, return inversely to the previous level and when the basic level is reached, i.e. the fictive level where you were before starting; the execution of the program, alternatively the block is terminated. Therefore the execute list is the most fundamental element in the Duet language and the understanding of the function of the execute list is consequently basic in order to sense the possibilities of the Duet.

Figure 3.3 illustrates the principle of the level oriented execution of a Duet block. We suppose that the instruction d1 has just been activated from a position called 'p'. To the left is the Duet program which is executed and the scheme rightmost shows how the Duet instructions are scanned.

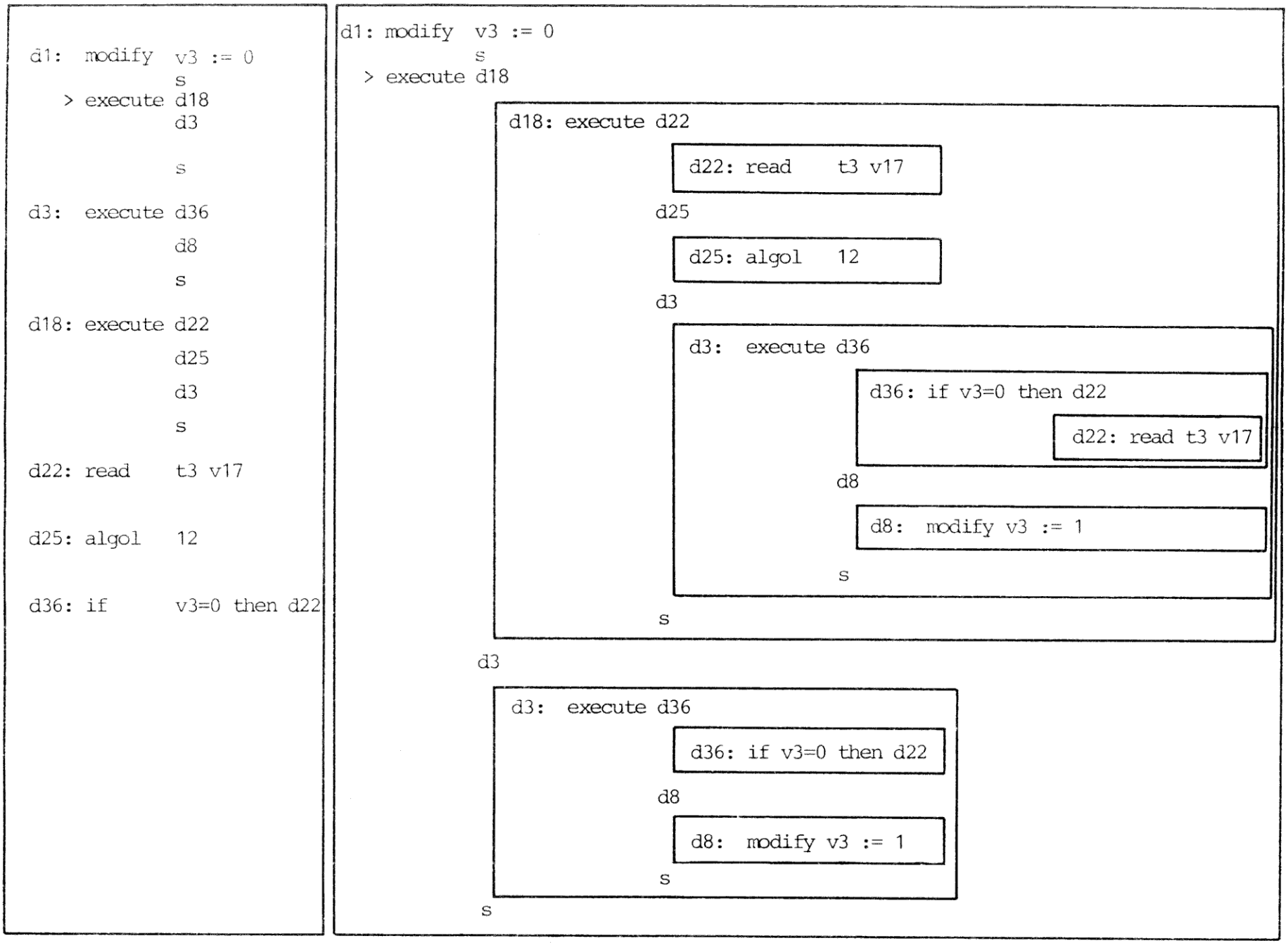


Figure 3.3: Execution of a piece of Duet program, starting in d1.

3.1.1 Duet Head

A Duet program consists, as shown in figure 3.1, of a Duet head and one or more Duet blocks.

The Duet head contains information for identifying the Duet program; used both when compiling and when running the compiled Duet program.

```
duet program 25/'teledata online'

localdata 23
```

Figure 3.4: Duet head

Duet program number and - name

Figure 3.4 shows the Duet head for a Duet program number = 25 and Duet program name = 'teledata online'; and this Duet program makes use of the local description no. 23. The Duet program name must not exceed 17 characters.

3.1.2 Duet Blocks

The Duet blocks have been introduced for two reasons. Partly to permit a local naming of the Duet instructions and partly as the basis of a user-/program controlled segmentation, which might be necessary if the program exceeds the 500 - 1000 Duet operations (2048 words of compiled program) which a Duet program, consisting of a single block, can contain.

Block number

All the Duet blocks in a program have a block number, which identifies the block unambiguously. This block number is used by references from an execute list to non local Duet instructions, that is, instructions in another block.

Entrypoint

As it is possible to compile each block seperately and as the Duet names are local within a block, such references to non local instructions are made by symbolic entrypoints, which are defined in the head of each block.

Each Duet block contains a block head, a number of Duet instructions, and a block end (cf. figure 3.1).

3.1.2 Duet Blocks

```
begin 4: user 35        ; block no. 4 belongs to
                        ; user 35
->          1 = d3
->          3 = d50
->         10 = d6

      duet instructions

end 4                   ; same no. as in the
                        ; block head
```

Figure 3.5: block head and block end.

In figure 3.5 you can see a block head and a block end for a Duet block with 3 entrypoints. Each entrypoint defines an entry number corresponding to a definite Duet name which must exist within this block.

The following value intervals apply to block number, user number, and entry number:

```
1  <=  block number <= 255
0  <=  user number  <= 127
1  <=  entry number <=  63
```

The definition of the user number (user <user number>) can be left out if the user number = 0.

More entry numbers can point to the same Duet instruction. Entry number 0 is always implicitly the first Duet instruction in the block and cannot be redefined.

3.1.2 Duet Blocks

Activation of the Duet block

When a Duet block is activated it is done, as mentioned above, by reference from an execute list in another block. The reference contains two pieces of information: a block number and an entry number. The Duet interpreter now investigates whether the indicated Duet block is already available in the core storage.

If the block is in the core storage, the Duet instruction, belonging to the indicated entry number, is activated at once. In this case, the activation of a Duet block does not take much more time than the activation of a local Duet instruction in the original block.

Duet array

If, however, the block is not present in the core storage it is automatically fetched from the Duet file to a Duet array the size of which has been defined by the control program.

If there are one or more sufficiently large, free sections in this array, the block is placed at the beginning of the smallest of these sections. Then everything proceeds as if the block had been there all the time.

If there, however, is no free section, the system must provide it by overwriting old Duet blocks in the array. All block references are counted in a blockcounter, which becomes the measurement of how desirable it is to avoid an overwriting. In the placing it is also attempted to keep the amount of these free minor places as low as possible in order to avoid an actual 'garbage collection'.

3.1.2 Duet Blocks

Check of block number

If the wanted block does not exist in the Duet file, the system will react with a Duet programming error (see section 5.3.2).

Check of user number

Before starting the execution of a Duet block, the Duet system checks whether the user number in the new block is legal compared to the old block:

From a block with user number 0 the program can activate any block, but once a block with user number <> 0 has been activated, this is stored as a user number for checking. After this, there are only allowed references to blocks with the same user number or with user number 0. The user number for checking can be neutralized (reset to zero) by letting the control program call the initialization procedure 'init_duetmaskine' (see section 5.2), after which it will be possible to process another user's blocks.

The purpose of this checking is to prevent a user adaption from executing another user's program.

Figure 3.6 illustrates this checking.

3.1.2 Duet Blocks

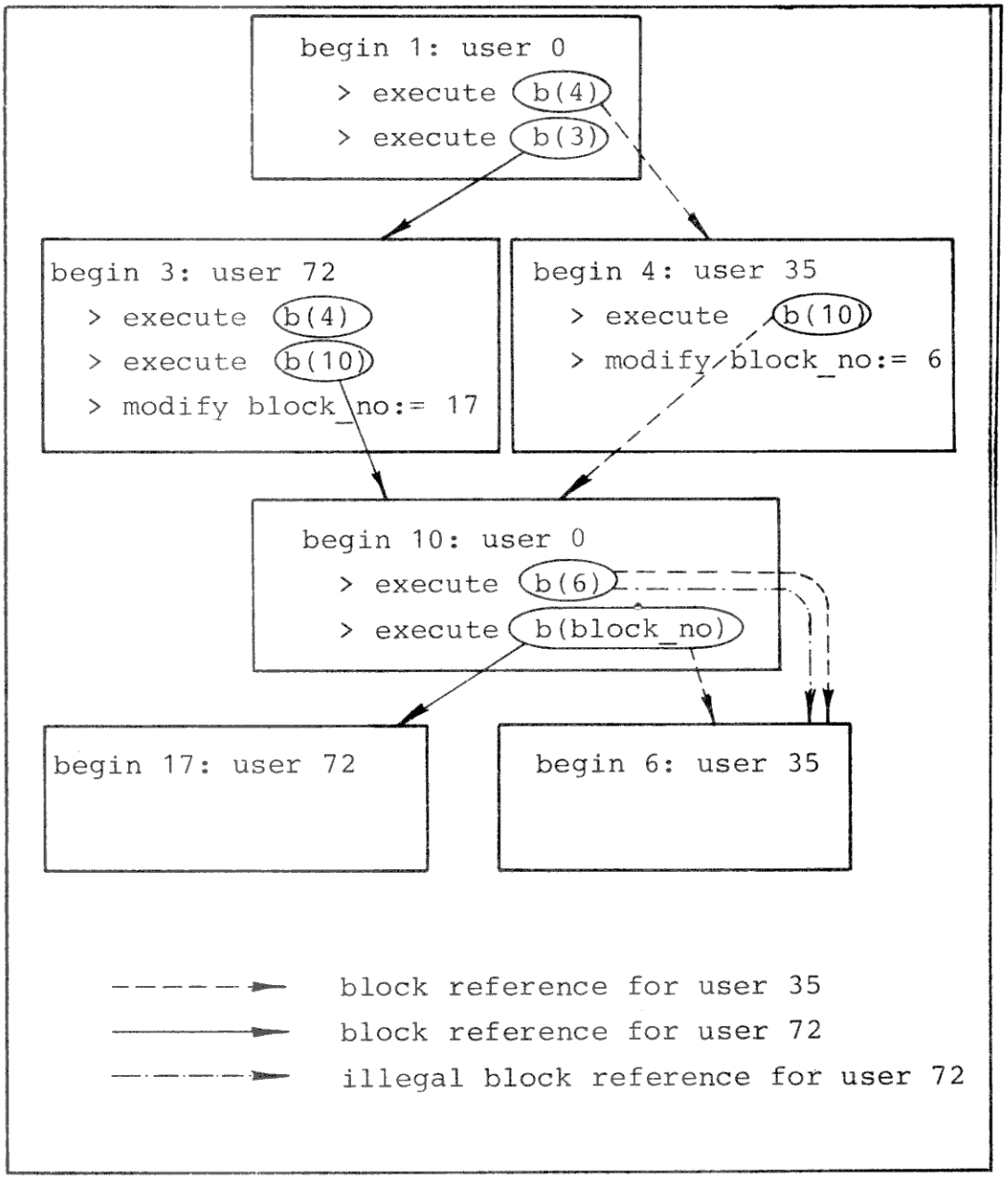


Figure 3.6: Checking user number in block references.

In the example, block 1 is a program block, common to all users. Block 10 is a common adaption block while the other blocks are individual user adaptions. It is not allowed to activate block 4 from block 3. It is legal to jump to block 6 from block 10 if block 10 has originally been called from block 3.

3.1.2 Duet Blocks

Return

When the selected Duet instruction in a block reference has been completed, the program will return to the previous block where it continues the processing of the execute list.

When a Duet program is activated the entry will be to the first Duet instruction in the block with the lowest number.

– altså ikke nødv vej 1

3.1.3 Duet Instructions

The format of a Duet instruction is shown rightmost in figure 3.1.

Duet name

The Duet instruction is identified by a Duet name which can be referenced from Duet operations in other instructions.

The Duet name is declared at the beginning of the first Duet operation in an instruction by the letter 'd', followed by an integer, and a colon (see figure 3.7).

```
d32: execute d33
             d17
             s
   > get     s 2
   > if      soda_result > 0     then d99
```

Figure 3.7: an example of a Duet instruction

A Duet name must be declared in the interval

d1 <= duetname <= d1023

and the compiler checks that the Duet names are unique within each block.

Duet operation

After the Duet name follows the Duet operation(s) that are included in the instruction. Each Duet operation must be terminated with a line feed and the start of the next operation is to be marked with the character '>'. An arbitrary number of Duet operations may be included in an instruction; in principle, it is possible to let the whole block consist of one long Duet instruction.

3.1.3 Duet Instructions

Each Duet operation consists of a Duet operator and one or more Duet operands, dependent on the type of the operation. The Duet operations are described systematically in section 3.2 and the possible operand types are described in the following section.

3.1.4 Duet Operands

The Duet operations may include different types of operands: variable references, constants, and Duet names which have all been described in this section.

Furthermore, references for record sets and record types are used by db operations, as it has been described in section 3.2.13.

Variable references

The Duet program recognizes and can refer to all the variables that are declared in their associated local data description (see references 2 and 3) without necessitating that field variable declarations have been generated for them.

Variable name

Variables can be referenced indiscriminately by name or by number. For a reference by name is used the full variable identifier from the variable declaration (note especially that also underlined space is important as distinct from what applies to the Algol programs).

Variable number

To a variable reference by number you only write the letter 'v' followed by the number of the variable. This method of reference can, however, not be used in Duet programs that are based on a data entry local data description where the variable numbers are anonymous.

Subscripted variable

If a variable is declared as an array, a subscription in the variable reference is usually required, as shown in figure 3.8.

3.1.4 Duet Operands

```
variable references by name:
      price_index
      item_price (price_index)
variable references by variable number
      v1            ; simple variable
      v25(2)        ; subscribed variable with
                    ; constant index
      v25(v1)       ; subscribed variable with
                    ; variable index
      v28(v25(v1)); subscription at several levels
```

Figure 3.8: variable references.

The subscript may be constant or a variable, which again may be subscripted in an arbitrary number of levels. If the subscript is a constant, an index check is executed during the compilation. When the subscript is a variable, the index check cannot be executed before the run time but the compiler checks the all index variables are declared without decimals.

Simple variable

A variable without an array specification is called a simple variable. Certain Duet operations demand that the referring variable is simple.

Variable type

Many Duet operations make demands on the variable type of a variable reference. In this manual, the following designations for variables with type restrictions are used, as shown in figure 3.9, below.

3.1.4 Duet Operands

variable designation	legal variable types
text var	text
numerical var (numvar)	word, long, real, date, result
word var	word, date, result
~~integer var~~	word var without decimal denotation
bit var	bits
rec no var	recno, result_recno

Figure 3.9: variable designations

Constants

The other type of Duet operands are constants in which the Duet language distinguishes between numerical constants, character constants, and text constants.

Numerical constants

Numerical constants are ordinary integers or decimal numbers. Decimal numbers can be stored by the compiler either as a floating point number (real) or as an integer with implicit decimals, dependent on the context in which the constant occurs.

Automatic normalization

The running Duet system will always provide automatic normalization, i.e. it will take into account the differences in the number of decimals between the operands to the left and right of an operator, cf. figure 3.10.

3.1.4 Duet Operands

```
> if  v17  =  100 then ....
> if  v17  =  100.000 then ....
```

Figure 3.10: automatic normalization

The two conditions in the figure are the same whether v17 is defined with 0, 1 or with 2 decimals.

Spill

However, in principle the Duet system does not check whether spill arises by variable assignment, or not. So users who want such a check, must compile the Duet system with spill.yes (cf. section 5).

Character constants

Character constants - or the so called short texts - are written as one, two or three characters enclosed in decimal points, as shown in figure 3.11. The characters may be letters, digits or special characters, but not decimal points. These characters are packed with their ISO values in 24 bits, right justified with possible zeroes to the left.

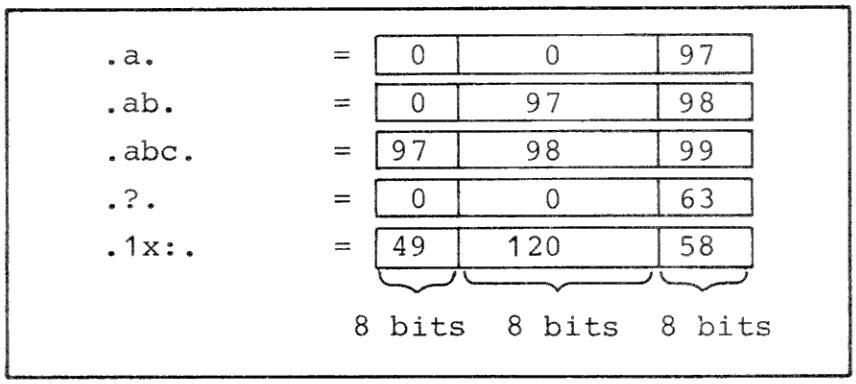


Figure 3.11: character constants.

3.1.4 Duet Operands

A character constant is per definition numerical and it can be used in all situations where numerical constants are permitted. The numerical constants mentioned in the manual include both numbers and character constants.

Text constants

Text constants are written as a text and bracketed by apostrophes, as shown in figure 3.12. Such a string can be empty and the upper limit of the amount of characters is only determined by how much a line can comprise.

It is stored like in Algol, i.e. justified left and terminated with as least one zero character.

```
'long text with special character?'
'short'
''              ; empty text
```

Figure 3.12: text constants.

All constants, referenced in a block, are stored together with the compiled Duet instructions so that the constants only take up space in the core storage as long as the current block is present in the Duet array (cf. section 3.1.2).

Duet names

Duet names make up the third type of Duet operands. The declaration of Duet names has been described in section 3.1.3, here it only concerns references to Duet names.

3.1.4 Duet Operands

A Duet operation can refer to another Duet instruction in the same block by indicating its Duet name.

```
> execute  d17
           d32
           s
> if       v35 = 1 then d0 else d33
```

Figure 3.13: Duet reference

The Duetabler compiler checks that all the referenced Duet names are declared within the block.

Apart from the declared Duet names there is a standard Duet instruction called d0. This is an empty instruction: nothing is executed when referring to d0.

Value elements

Finally; in the Duet language the concept, value element, exists which covers a combination of the mentioned types of operands.

Numerical value element

A numerical value element is thus either a numerical constant, a character constant, or a numerical variable (simple or array element).

Text value element

A text value element is either a text constant or a text variable (simple or array element).

3.2 Duet Operations

In the following section, the separate possible Duet operations are described systematically. Below is an outline of all the Duet operations, classified according to their functions:

Executing operation	execute
Variable assignment	modify
	compute
	assign
Conditional operations	if-then-else
	case-of
	action-of
Repeating operations	for-do
	while-do
Reading operations	getline
	read
Printing operations	print
Database operations	get
	next
	lookup
	create
	put
	delete
	newset
Other operations	select
	exit
	algol

3.2.1 Execute

The fundamental executing Duet operation is the execute list whose function has been described in section 3.1.

```
d7: execute   d123              ; local duetreference
              d8, d36, d15
              b (16,5)          ; block reference
              b (adp_block,2)   ;       -    , var.blockno
              b (v15,v16)       ;       -    , var.blockno
                                ;              and entryno
              b (18), d4        ;       -    , without entryno
```

Figure 3.14: Execute list

In figure 3.14 the format of an execute list is shown. It is initiated with the operator 'execute' and terminated with a Duet stop; the character 's'. Between these, an arbitrary number of action references can be placed; either more in one line, separated by commas or one in each line without any comma.

An action reference can either be a local Duet reference or a block reference.

Local Duet reference

A local Duet reference is only a Duet name ('d' followed by an integer), i.e. a reference to another Duet instruction in the same block (see section 3.1.3).

When executing the execute list, a local Duet reference will cause a temporary transfer of the control to the referred Duet instruction.

3.2.1 Execute

But the position in the current execute list is stored and when the instruction has been executed, the control is returned, so that the next reference in the list can be executed.

Block reference

A block reference is used for execution of a Duet instruction in another block. A block reference is indicated with a 'b', followed by a block number and an entry number in brackets. Both the block number and the entry number can be indicated as an integer constant or a variable reference. In the latter case, the contents of the indicated variable at the run time, determines which block/entry is being activated.

The entry number can be left out in the block reference; in that case the first Duet instruction in the block is implicit.

Note, that the block reference only can occur in an execute list. All other Duet operations can only refer to local Duet names.

The compiler cannot check whether a block reference is legal or not. Partly because the single blocks are compiled independent of each other so that no checks are executed across the block limits. Consequently, the block reference must be checked by the Duet system at the run time, as it has been described in section 3.1.2.

When reaching the Duet stop in the execution of an execute list, this is finished. It is then investigated whether there are more operations in the current instruction and if that is the case, the execution of these are continued. If not, a return to the previous level in the dynamic execution hierarchy is performed (cf. section 3.1).

3.2.1.2 Program Points

A Goto operation cannot be expressed in Duet. It contradicts the hierarchical execution of the Duet program's execute lists, whose principle is that when an instruction has been executed in its outmost consequence, the program will return to the previous level. The execution of an instruction may include transfer of the control to new levels but if the program does not end in an infinite loop it will, sooner or later, return to the original instruction.

In certain situations, however, one may need to short-circuit the execution and return directly to one of the previous levels, e.g. by data errors; or to exit from a program loop before its stopvalue has been reached.

This can be done by the exit operation in connection with a program point in the execute list.

```
> execute      d3
               d8
               d13
    p2:        d13, d22
               d36
               s
```

Figure 3.14: Execute list with a program point.

Program point declaration

A program point is declared at the beginning of a line in an execute list with the letter 'p' followed by an integer in the interval 1-9, and a colon.

3.2.1.2 Program Points

When an execute list is executed and a line with a program point declaration is to be processed, a marking of the specified program point is stacked together with the pointer for the current position in the list. This makes it an active program point.

Active program point

At any time there can thus be several active program points and there can, at one and the same time, be several (program points) with the same number. To each active program point a level is attached, which corresponds to the dynamic level in the program at which the corresponding execute list has been activated (corresponding to its current position in the stack).

Returning from the last instruction in the line, to which a program point has been attached, this is made passive again in connection with the unstacking. In the example, p2, at the current level, will only be active as long as d13 or d22 or instructions activated from them are being executed.

If d13 is activated from somewhere else, p2 will not become active (unless p2 has also been declared a program point in the activating operation).

Exit

The exit operation (described in section 3.2.15) can refer to a program point. When the exit is performed, the Duet program will return to the nearest active program point (relative to the current level).

From this point, the next line of the execute list is continued without any regard to whether the previous line or the execute lists activated on the intermediate levels have been concluded or not.

3.2.1.2 Program Points

If there is no active program point with the stated number, the interpreter returns to the basic level, i.e. the Duet program is terminated.

Select

The operator 'select' (see section 3.2.14) can define an automatic exit to a program point when data errors, programming errors and/or system errors occur. Different kinds of program points can be selected for different kinds of errors.

3.2.1.3 Conditional Compilation

The execute list can be supplied with a suppression specification which causes a conditional compilation.

```
> execute     d35
    t1 p7:    d123, d8, b(18)
              d32
              d13
    t2        b (block_no, entry_no)
              s
```

Figure 3.16: Execute list with suppression specifications.

Suppression specification

A suppression specification is indicated by the letter 't' (for test) followed by an integer in the interval 1-9.

In a normal compilation such a line will be suppressed and so it is not included in the compiled program. But if an 'include' parameter is specified for the compilation of the Duet program (cf. section 4.2) the line will be compiled if the t-value in the line is less than the suppress value of the include parameter.

3.2.1.3 Conditional Compilation

As shown in the example, you can also suppress a line where a program point has been declared. In this case the program point will never become active.

Neither the suppression specification nor the program point declaration can be written in the same line as the execute operation. If these are wanted for the first reference in the execute list, this reference must be specified on a new line, as shown in figure 3.17.

```
> execute
    p8:    d32
           d33
           s
```

Figure 3.17: Program point at the 1 st Duet reference.

3.2.2 Modify

The modify operation can assign variables of all kinds by a simple transfer of a value to a variable. Furthermore, the operation can be used for accumulation in numerical variables. However, no execution of calculations can take place, and assignment is performed without the use of working locations, thus making modify the fastest kind of assignment.

The format of the modify operation is a list with a variable length, terminated with a Duet stop. Figure 3.18 shows some of the possibilities in a modify list.

```
> modify      index := 3
              arrayvar(index) := simple_word
              counter :+ 1
              saldo   :- payment

              textvar := 'text constant'

              array2  := array1
              array   := 0
              s
```

Figure 3.18: Modify list

Each line consists of a left side, an assign operator, and a right side.

Left side: var.ref.

The left side is normally a reference to a simple variable or to an element of an array variable. In the latter case the index must be an integer constant or a simple integer variable (cf. section 3.1.4).

3.2.2 Modify

Consequently, subscription is not possible in more than one level at the left side.

Assign operator

The assign operator can be := for ordinary assign and :+ or :- for accumulation.

Right side: value element

The right side must be a value element, i.e. a constant or a variable reference. At the right side one may subscribe at several levels and a sequence of special operators are available, which have been described in the following subsections.

The right side must correspond to the left side, as regards types. In this way a numerical value element can only be assigned to a numerical variable and a text element only to a text variable.

The following detailled description of the modify operator can be divided into groups according to the kind of the right side:

1. numerical assign
2. text assign
3. anonymous assign
4. special assign

3.2.2.1 Numerical Assign

Figure 3.19 shows the possible kinds of numerical assign.

```
>modify     index := 3
            counter :+ 1
            saldo   :- payment
            array(index) := simple_var
            code := .xy.
            item_group := digits (6,4) of item_no
            rest := item_no mod item_group
            kind := item_no // item_group
            s
```

Figure 3.19: Numerical assigns

For numerical assign, all three assign operators can be used with the following meaning:

:=	numvar := ritht side
:+	numvar := numvar + right side
:-	numvar := numvar - right side

The left side must be a numerical variable reference, i.e. a reference to a simple variable or an array element of the type: word, long, real, date or result.

The right side may be a numerical constant, a character constant or a numerical variable reference.

Digits

Furthermore, the operator 'digits' can be used at the right side. Its function is illustrated in figure 3.20.

3.2.2.1 Numerical Assign

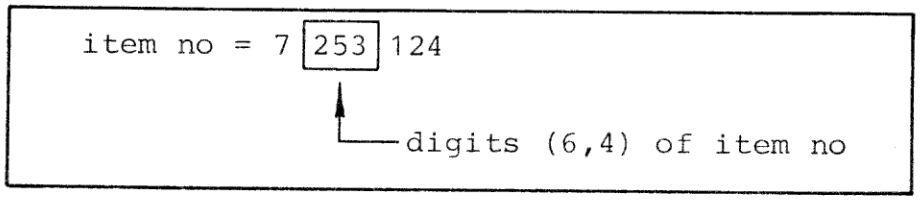


Figure 3.20: digits

The digits operator extracts a group of digits from a variable. The variable is regarded as a decimal number, in which the digits are numbered from the right so that the least significant digit gets the number 1. The result of

digits (a,b) of c

where a>=b, is an integer consisting of the digits from the digit position 'a' to the digit position 'b' (both included). The result of the operation in figure 3.20 is thus 253.

Note: no regards are taken to implicit decimals in the variable on which 'digits' operates.

The result of

digits (2,1) of balance

where balance is declared with 2 decimals will then be these decimals.

Integer devision, mod

Finally the right side can be supplied with an operator for integer division (//) or rest calculation (mod). These binary operators can operate on integer constants and/or integer variables and they also demand the left side variable to be of integer type ('word' or 'long' without any decimal indication), see figure 3.20b.

3.2.2.1 Numerical Assign

```
>modify  rest := 12 mod index
         day  := date mod 100
         year := date // 10000
         s
```

Figure 3.20b: mod and //

Automatic normalization

Numerical assign provide for automatic normalization, i.e. differences in the number of decimals between the left side and the resultant right side and differences in type are automatically taken care of, (except for integer division and modulo calculation).

```
>modify  w_2_dec   := 123.4
         w_integer := 123.4
         w_var     := l_var
         w_var     := r_var
         s
```

Figure 3.21

In the example, figure 3.21, w_2_dec (declared with 2 decimals) will, after the operation, have the value 12340 (which means 123.40) while the w_integer contains 123.

It is allowed to assign a 'long value' or a 'real value' to a word variable but the interpreter does not check whether the variable can contain the value or not (cf. section 3.1.4).

3.2.2.2 Text Assign

```
>modify address    := 'falkoner alle 90'
        txtvar(2)  := txtvar(1)
        var_name   := name (v17)
        var_name   := name (var(var_no))
        s
```

Figure 3.22: Text assign

For a text assign only the assign operator := can be used and the left side must be a reference to a variable of the type 'text' (simple or array element).

The right side in the text assign can either be a text constant or a reference to a text variable.

name

Furthermore, the operator 'name' can be used at the right side. This operator delivers, as a result, the name of the specified variable in accordance with the language code used for the ld-compilation. This variable can be specified directly as in

```
name (v17)
```

which gives the name of the variable v17.

name (var)

The variable can also be specified indirectly by the operator 'var', e.g.

```
name (var (v32))
```

Here, v32 contains the variable number of the variable whose name is wanted.

3.2.2.2 Text Assign

The compiler checks that the left side variable is large enough to contain the resulting text at the right side.

3.2.2.3 Anonymous Assign

```
>modify sort_crit := var (user_key)
        s
```

Figure 3.23: Anonymous assign

The anonymous assign can be executed by means of 'var' specified as the first operator at the right side. In the example, figure 3.23, the variable 'user_key' must contain a variable numer which indicates the variable to be assigned to 'sort_crit'.

The left side must be numerical (and should be a 'long'). As the compiler cannot check the type of the resulting right side, the following conventions apply to the assignment:

result.right side		
word long real date	are delivered as a numerical value with regard to the receiving variable's type and number of decimals	
text	6 first characters	are delivered as a bit pattern without any decimal conversion
bits	4 first bytes	
recno	2 bytes	

If the appointed right side variable is an array variable, its first element is used according to the same rules.

3.2.2.4 Special Assign

Special assign means an assign to something which is not a single numerical variable or a text variable

That concerns

- a transfer of a complete array
- a reset to zero of a complete numerical array
- an assign to a bit variable and
- an assign to a recno variable

```
>modify array_1      := array_2
       key_aggr      := record_key
       saved_recno   := recno_result
       array_2       := 0
       s
```

Figure 3.24: Special assign

The assign operator in special assign can only be :=.

Transfer of array

For transfer of a whole array you must write an array identifier at both sides of the assign operator, without any indication of the subscript.

In this type of moving, an absolute type correspondence between the left - and right side is demanded. For numerical values it is also necessary that the two arrays have been declared with the same number of decimals, whereas it is required that the length of the single elements are identical in both arrays, concerning arrays of the type text/bits.

3.2.2.4 Special Assign

If the arrays have different lengths only as many elements as defined by the shortest array are transferred. Finally no more than 2047 words can be transferred altogether.

The numerical array reset to zero

The resetting to zero of a complete array can be done by writing an array identifier without a subscript at the left side and the number 0 at the right side. However, this possibility only concerns numerical arrays.

Assign bit variables

A bit variable can be transferred to another bit variable if the two variables are of the same length.

Assign recno variables

Finally a variable of the type recno can be assigned either with the value of another recno variable or with the number zero.

recno: 0 eller en annen recno

3.2.3 Compute

The compute operator can be used for the assignment to numerical variables of the value of general numerical expressions employing the ordinary numerical operators and brackets.

The value of such a numerical expression can be assigned to several variables, as in Algol. However, 'compute' cannot be used for text assign.

The format of the compute operation is a list of variable length terminated with Duet stop (the character 's'), see figure 3.25.

```
>compute v1 := v12(3) := 5
         saldo := balance - payment(ser_no)
         price(price_code+2) := 22*(v13(v1)+v17)
         s
```

Figure 3.25: Compute list.

Each line, which constitutes an assign in itself, consists of a left side, an assign symbol (:=) and a numerical expression at the right side.

Left side: general var.ref.

The left side is a reference to one or more simple numerical variables or to elements of numerical arrays. The subscript of an array element can here - as the only place in the Duet language - be a general numerical expression according to the same rules that apply to the right side. If more variable references appear on the left side, these are seperated by the assign symbol.

3.2.3 Compute

Right side: numerical expression

The right side is a numerical expression. References to numerical variables (simple or array elements, as on the left side), numerical constants (integers and decimal numbers) and character constants can appear as operands. The operands are separated by the usual numerical operators,

+ - * /

and by brackets.

The right side may degenerate into one numerical variable or constant but in these cases it is normally cheaper to use the modify operation.

Real working register

All calculations in a compute list are performed by means of a real working register with usual regard to possible decimals in the operands. However, if any of the operands are long values with more significant digits than 36 bits can contain, the computation will cause a loss of accuracy.

Priorities

The calculation of an expression in the compute list is carried out according to the following (normal) priority rules:

1) subexpressions in brackets before the surrounding expressions
2) operations with * and / before operations with + and -
3) from left to right.

3.2.4 Assign - of

Selective assign

This operation, called selective assign, is used to assign a simple integer variable governed by the value of a test variable.

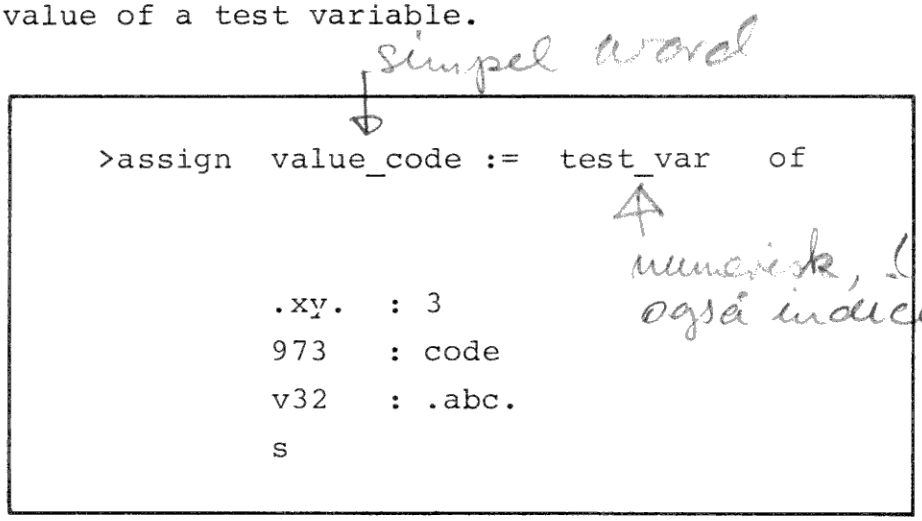


Figure 3.26: Selective assign

Assign list

The operation contains a (variable length) assign list of test values (to the left of :) and their corresponding assign values, see figure 3.26. Both test values may be simple integer variables, integer constants or character constants but none of them must exceed 24 bits.

Assign variable

The variable that is to be assigned must be a simple integer variable. The test variable, however, may be an arbitrary numerical variable, simple or array element; only it must not be of the type 'real' or have a decimal indication. The assign variable and the test variable may be the same variable.

The assign list may either be terminated with Duet stop (as in figure 3.26) or with an alternative value or an alternative action (figures 3.27 and 3.28).

3.2.4 Assign - of

```
>assign  v2 := v1 of
         1  :  100
         2  :  0
         else  v35
```

Figure 3.27: Selective assign with an alternative value

```
>assign type := code of
        .ov. : 1
        .ok. : 2
        .ok2.: 2
        else   d19    ;   error action
```

Figure 3.28: selective assign with an alternative action

The operation works in the following way: The value of the test variable is successively compared with the test values in the list. When meeting a test value which is equal to the value of the test variable the comparisons are stopped and the assign variable is given the corresponding assign value.

If none of the test values apply to the testvariable, the reaction depends on whether the assign list is terminated with a Duet stop, an alternative value or an alternative action.

3.2.4 Assign - of

Alternative value

An alternative value is stated with 'else' followed by a simple integer variable, an integer constant or a character constant; and this value is always assigned to the assign variable if none of the test values fitted.

Alternative action

If an alternative action is stated, i.e. 'else' followed by a Duet name, this Duet instruction will be executed if no test value was found.

Duet stop

Finally, the Duet stop can be stated. This corresponds to an empty alternative action like 'else d0'. In this case nothing is executed.

3.2.5 If - then

A conditional execution of a Duet instruction can be effectuated by the if - then operation, whose format is shown in the examples in figure 3.29.

```
>if v32(v14(2)) <>0 then d19 else d20

>if balance < credit_maximum then d930

>if code = .t13. then d14 else 365
```

Figure 3.29: if - then operations.

Numerical relation

Between 'if' and 'then' a single numerical relation must be stated, in which the symbols below can be used as relation operators:

- < : smaller than
- > : greater than
- = : equal to
- <>: different from

No compound symbols can be used (i.e. <=) neither any compound relations (i.e. 'and' and 'or'). The left side of a relation must be a numerical variable reference (simple or array element). The right side may furthermore be a numerical constant or a character constant; that is, a usual numerical value element.

If the condition is fulfilled (the relation is true), the Duet instruction stated after 'then' is executed. If the condition is not fulfilled, the Duet instruction stated after 'else' is executed.

If 'else' is left out, nothing is executed in the 'else' situation, corresponding to 'else d0'.

3.2.6 Case - of

The case branching in Duet resembles, in its functions, the Algol case statement.

```
> case   test_var of
     1:  d32
         d318
         d0
     4:  d32
         else d27
```

Figure 3.30: case operation

The format of a case operation is shown in figure 3.30. The numbers in front of the Duet names are redundant comments. They can be left out but if a number is stated, the compiler checks whether it fits its location in the list.

The test variable can be an arbitrary numerical variable, simple or array element; only it must neither be of type 'real' nor be declared with decimals.

In the execution of a case operation the Duet instruction is activated, whose location in the list of Duet names corresponds to the value of the test variable.

If the value of the test variable is less than 1 or larger than the amount of Duet names in the list, the instruction after 'else' is always executed.

The case list may be terminated with the Duet stop ('s') instead of 'else', which corresponds to 'else d0'.

3.2.7 Action - of

Selective branching

This operation, called selective branching, resembles, in principle, the selective assign (section 3.2.4). It causes an execution of a Duet instruction, conditioned by the value of a test variable.

Where the range of the case branching (cf. section 3.2.6) is limited to a sequence of consequtive positive integers from one and upwards, then the range in selective branching is almost unlimited. However, a search must be made in a list of the single relevant values. The current situation must then determine which, of these two possibilities, is the most advantageous.

MAX LANGUE?

```
> action  text_var      of
          .xy.     :    d19
          973      :    d318
          v12      :    d0
          else          d27
```

Figure 3.31: selective branching.

The operation contains a (variable length) action list of testvalues and corresponding Duet actions. The list is terminated with Duet stop ('s') or an alternative action (as in figure 3.31).

The test variable can be an arbitrary numerical variable, simple or array element, only it must not be of the type 'real' or be declared with decimals. Simple integer variables, integer constants and character constants can be used as test values. The test values must not exceed 24 bits.

3.2.7. Action - of

In the execution of an action-operation, the test variable is compared successively with the test values of the list. If a test value is found which equals the value of the test variable, the comparisons are stopped and the corresponding Duet instruction is activated.

If none of the test values match the test variable, the operation is ineffective unless an alternative action has been stated with 'else'. In this case the instruction, stated there, is executed.

3.2.8 For - do

Program loops are made in the Duet language by the 'for - do' operation or by the 'while' operation (section 3.2.9).

```
> for control_var := start, stop do d32

> for v17 := 3,5 do d100
```

Figure 3.32: for - do operations

The format of a for - loop is shown by the examples in figure 3.32.

The control variable must be a simple integer variable while the start - and stop value may be integer numerical value elements.

The operation is executed in the same way as the Algol sentence

```
for control_var := start step 1 until stop do....;
```

only with the difference: that in Duet, the stop value is computed once and for all before starting the loop. It is thus not possible to interrupt the loop from within by changing the value of the stop element.

An interruption of the loop from within can, however, be caused by changing the value of the control variable or by the exit operation (see section 3.2.1.2 and 3.2.15).

3.2.9 While - do

A while - loop in the Duet has the format, shown in figure 3.33.

```
> while testvar <>0 do d313
```

Figure 3.33: while operation

A numerical relation must be indicated between 'while' and 'do' and it must fulfil the same syntactical demands as the relation in the if - then operation (cf. section 3.2.5).

The operation works in the following way:

If the relation is true, the stated Duet instruction is executed and the relation is re-tested. When the relation gives the result: false, the Duet instruction is not executed and this terminates the operation.

3.2.10 Getline

read_general

The 'getline' operation causes a call of the reading procedure 'read_general' which reads a text string from the zone 'readz'. (This zone must be opened by the control program to the input text area).

```
> getline textvar
```

Figure 3.34 getline

The parameter to the getline operation is a text variable in which all of the read text string is delivered. Furthermore, the text is separated into fields to be used by the reading operation 'read' (cf. section 3.2.11).

End of string

Getline terminates the reading when meeting a character in the input, which, in the system's character set table has been defined as an 'end of string' character (see section 3.2.11). This is normally the characters line_feed (ISO value 10, nl) and end_medium (ISO value 25, em).

If there is no 'end of string' character the reading will terminate after 150 characters, corresponding to the longest line 'read_general' can read.

If the text variable stated as parameter for 'getline' is too small to contain the whole input line, only part of it will be transferred to the variable. Yet the whole line is always available for interpretation by the Duet operation 'read'.

3.2.10 Getline

In certain program types it is more appropriate to let the application program activate the reading of text strings by calling the 'read_general' procedure. In this case, the corresponding Duet program must not use the getline operation.

3.2.11 Read

A text string delivered by the getline operation or 'read_general' can be looked upon as a sequence of fields which can be recognized successively by the Duet operation 'read'.

The read operation can

- read a field from the input string (numerical field, text field or character field) and

- perform a value check of the field.

or

- assign a field's standard value.

3.2.11.1 Classification of fields

Character set table

The interpretation of a line is controlled by a character set table which is defined by the Duet system's initialization procedure 'init duet2', (but may be redefined by the control program). For each ISO value the table determines a character class and an internal character value.

The internal character value is equal to the ISO value in the standard table, except for capital letters (ISO value 65 - 93) which are altered to small letters (97 - 125).

The character class is used for determining the termination of the text string, as well as the limits between the fields and the kind of the fields in the test string. Figure 3.35 shows the existing character classes and their standard initialization.

3.2.11.1 Classification of fields

order	meaning	character
<9	illegal	
9	blind	
10	end of string	nl em (new line/end medium)
11	text delimiter	' (apostrophe)
12	comment delimiter	" (quotation marks)
13	digit	0 1 2 3 4 5 6 7 8 9
14	ordinary delimiter	sp , (space and comma)
15	sign/standard mark	- (minus)
16	decimal point	. (point)
>16	other characters	letters, special characters

Figure 3.35: character classes

Comment field — A comment field is a number of characters given between two comment delimiters (quotation marks). 'Read' always skips any comment fields.

Text fields — A text field is a number of characters stated between two text delimiters (apostrophes).

Numerical field — A numerical field consists of digits, possibly with a decimal point and/or a prepositioned minus character. A post positioned minus character cannot be read.

Character field — A character field consists of at most three single characters (a letter or a special character followed by letters, special characters and/or digits). These characters are packed as a character constant by 'read'.

3.2.11.1 Classification of fields

end of string

The end of string character (nl/em) cannot be included in any field. This means that if the terminating text-/comment delimiter is missing, the text/comment is terminated by an end of string character.

Ordinary delimiter

Ordinary delimiters (sp/,) may be included in text fields and comment fields but they terminate numerical fields and character fields.

3.2.11.2 Read Specifications

The 'read' operation is controlled by a read specification which contains a read mode and a reading variable. For each call of 'read', one field of the input string is interpreted and the value of this field is transferred to the reading variable if it matches the stated read mode.

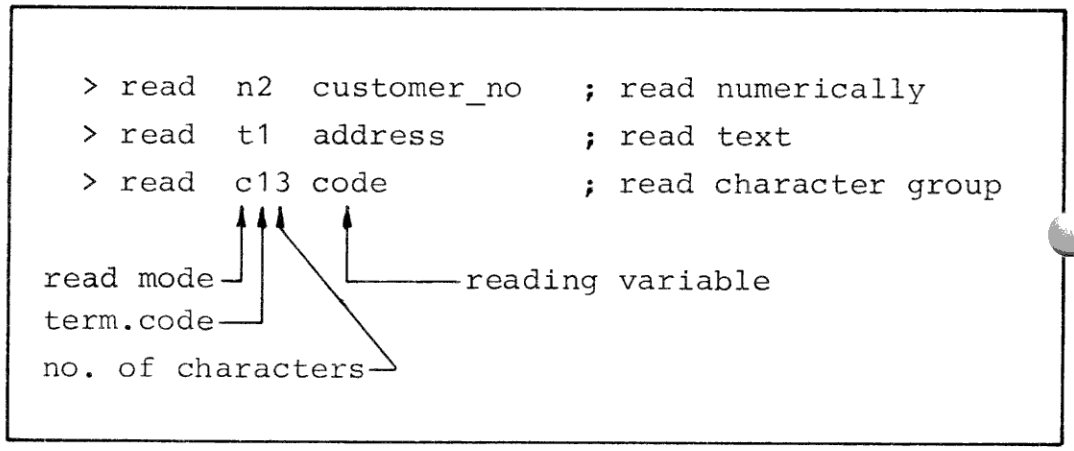


Figure 3.36: read

Read mode

The read mode indicates the nature of the expected field and the way in which it is to be read and be represented in the reading variable.

3.2.11.2 Read Specifications

n
numerical

n indicates the reading of a numerical field, i.e. an integer or a decimal number with a sign, if any. The reading variable must be numerical and the value is delivered in accordance with the variable type and number of decimals.

t
text

t stands for the reading of a text field with surrounding text delimiters (apostrophes). In the reading variable, which must be of the type 'text', the characters between the two text delimiters are delivered in a normal text form.

c
chars

c indicates the reading of a character field (chars). For this purpose, the maximum number of characters allowed in the field, must be specified (1, 2 or 3). The reading variable must be an integer variable (word) in which the characters are represented in the same way as a character constant in the Duet program (see section 3.1.4). A field starting with a digit cannot be read as a character field.

Terminator code

The terminator code defines which type of delimiter is allowed as terminator for the current field. The following codes apply:

0: the terminator is not read and is not checked.

1: the terminator must be an ordinary delimiter (char.class 14).

2: the terminator must be an end of string delimiter (char.class 10).

3: the terminator may be an ordinary - or an end of string delimiter.

3.2.11.2 Read Specifications

Resultvar readterm

After the reading operation the result variable 'result.readterm' states which terminator was read:

result.readterm = 0: no terminator read
> 0: ISO value for terminator.

If the terminator is read, all the coherent terminators are read up to the next field or perhaps up to the end of string. In this case only the last terminator is stored and checked.

Several read spec

Several read specifications can be stated in one 'read' operation; by which is meant that several different kinds are allowed for the current input field, see figure 3.37.

```
> read  n1 text_no, t1 text, c13 code
```

Figure 3.37: 'read' operation with several read specifications

In the execution of a read operation it is checked if the input field fulfils the syntactical demands, which corresponds to the read mode in the first read specification. If this is not the case, a reading is attempted in accordance with a possible additional read specification in the current operation. 'Read' continues in this way, until the field has been found syntactically correct, or there are no more read specifications. In the latter case an error reaction is activated by calling the procedure: 'duet_data_fejl' (cf. section 5.3.1).

3.2.11.2 Read Specifications

```
> read    s,   c13 line_code
```

Figure 3.38: 'read' specification with reading of a prepositioned delimiter

In front of the first read specification you can state 's'. This means that if an ordinary delimiter (SP/,), which has not yet been read, is positioned in the current location of the input string, then the following field is not read.

This marking is only significant when reading the first field of the line or if the 'read' operation, last executed was reading with the terminator code 0.

3.2.11.3 Check of value spectrum

When a field has been found syntactically correct, a possible value check is executed, determined by the value spectrum of the reading variable in the ld-description. It is checked that the value of a numerical variable is inside its permitted value spectrum. The number of characters read to a text variable is checked to be inside the permitted interval.

For variables without any value limits the only check is that the value can be contained in the current variable.

3.2.11.3 Check of value spectrum

If the read value is outside the permitted value spectrum an error reaction is activated.

3.2.11.4 Standard Value

Explicit standard mark

If only a single character of the class standard mark (minus character) is given as a field in the input string, no actual reading is executed. Instead, a standard value is marked for the first specified reading variable if this is declared with a standard value in the ld-description. If not, an error reaction is activated.

Implicit standard mark

The same possibility of an automatic standard value marking exists if you continue reading of fields, after the input string has been exhausted. This will be regarded as if a standard value had been stated for the field.

This facility makes it possible (without any special syntactical marking) to shorten those lines in the input, in which all the remaining fields should receive a standard value.

stdassign

The processing of the standard mark depends on the value of the variable 'stdassign':

- stdassign <> 0: the standard value specified for the read variable in the ld-description is assigned to the variable.

- stdassign = 0: the value of the reading variable is not altered as a standard value is supposed to have been assigned to the variable previously, e.g. by a db-operation (cf. section 3.2.13).

3.2.11.4 Standard Value

The variable 'stdassign' is reset to zero by the call of 'init_duetmaskine'. It may be changed by the control program after this call but it may also be changed dynamically from the Duet program by using the 'select' operation (see section 3.2.14).

Resultvar

After the reading opertion, the result variable 'result.readspec' specifies, which read specification has been executed:

result.readspec = 0: standard value or prepositioned delimiter read

> 0: the number of the effectuated read specification

(If the result variable has not been defined, this piece of information will simply disappear).

3.2.12 Print

'primula'

The print operation is used for printing results (by means of the 'primula' system). The printing is carried out via a line buffer of 132 positions, which may be assigned in an arbitrary sequence.

By a special print command this line buffer is transferred to a result area via a zone in the zone array 'prinz'. The control program defines how many zones this zone array contains and opens each zone to its respective result area (cf. section 5.2.1 and 5.5.1).

With the 'select' operation (which has been described in section 3.2.14 the current zone within the zone array is selected and this choice is valid until the next call of 'select'. The call of the 'init_duetmaskine' procedure always selects prinz(1) as print channel.

The format of the print operation is a list of variable length, terminated with a Duet stop (the character 's') as shown in the example, figure 3.39.

```
> print <p 1 t23>    : cust_name; form_feed
        <25 n8>      : cust_no
         d335        :           ; compute balance
        <21 25 n6.2>: balance
        < 1 25 c9>   : .=.
        < 1       >
        s
```

Figure 3.39: print list

3.2.12 Print

Each line is either a Duet name or a print line consisting of a layout and a value specification.

Duet name

If the line is a Duet name it must be the name of a Duet instruction in the same block and this Duet action will be executed in the same way as in an execute list. No block references can be written. It is not allowed to write more than one Duet name in one line.

Print line

A print line consists of a layout specification which describes how the printing is to proceed and a value specification which states what is to be printed.

3.2.12.1 Layout Specification

The layout is surrounded by brackets <>. Between these is stated either:

- a solitary horizontal specification

or

- a vertical specification

and/or

- a position specification
- a layout type and
- a layout parameter

Horizontal specification: basic position

A horizontal specification is stated with the character 'h' followed by a position indication which may be a numerical constant or a single integer variable in brackets. Hereby a basic position is specified which, during the run, is added to all the position values until meeting a new horizontal specification.

3.2.12.1 Layout specification

The basic position is reset to zero by the 'init_ duetmaskine' procedure.

Vertical specification

A vertical specification indicates that the line buffer must be transferred to the result area, after which the printing of either a form feed, a vertical tabulation or a line feed (ISO value, 12, 11 or 10, respectively) proceeds. This is stated in the layout with the character p, w or b, respectively - the latter may be initiated with a parameter that specifies the number of line feeds. The parameter may be an integer constant or a simple integer variable. In the latter case, the current number of line feeds are not determined until during the run. The numer of line feeds must not exceed 70, corresponding to a full page A4-vertical.

Position

The position indication determines the number in the line buffer on the first character of the current field. The indication may - like the horizontal specification - be either an integer constant or a simple integer variable in brackets. The value of the position must be inside the range 1 - 127.

Layout type and layout parameter

The layout type and the layout parameter determine the format and the size of the field to be printed in the linebuffer; and they also make certain demands on the type of the value indicated by the value specification.

t: text

The layout type t indicates text printing and the value must be of the type 'text'. The layout parameter, which may be an integer constant or a simple integer variable in brackets, indicates how many positions are available in the line buffer. If the text is more extensive, it is shortened.

3.2.12.1 Layout Specification

c: char

The layout type c indicates the character printing (charprint). The value must be numerical and it will be interpreted like a character constant (cf. section 3.1.4). The layout parameter must also here be an integer or a simple integer variable in brackets which indicates how many times the character constant must be printed. (The character printing is thus suited for e.g. printing of underlinings and the like). If the character constant consists of more than one character the whole character sequence will be repeated.

n: num

The layout type n indicates a numerical printing and demands the value to be numerical. The layout parameter for this, is constructed by the following elements, in the sequence shown below:

- a fixed-sign marking
- a zero representation statement
- a zero value statement
- number of principals
- number of decimals

fixed sign

The fixed sign marking, indicated by a minus character, means that the sign of a possible negative value always must be printed in the first field position. If the marking is left out, negative values are printed with floating-point signs, i.e. immediately before the first significant principal digit.

zero repr.
z *

The zero representation statement specifies how insignificant figures are to be printed. If the character 'z' is stated, prepositioned zeroes are printed and if '*' is stated, this character is printed instead of prepositioned zeroes.

3.2.12.1 Layout Specification

If the statement is left out, the prepositioned zeroes are printed as blanks.

zero value
b

The zero value statement specifies, with the character 'b', that the value zero should be printed as a totally blank field, without regard to a zero representation statement, if any.

Principals
number

The number of principals is expressed with an integer stating how many character positions are, at most, to be included in the principal part of the number. If a floating-point sign is used the minus character will take up one of these positions. The principal specification is the only parameter element which cannot be left out.

Decimals
.number

The number of decimals is specified with a point followed by a number indicating the number of decimal digits to be printed. These will always be printed in full.

Figure 3.40 gives an informal outline of the layout possibilities:

3.2.12.1 Layout Specification

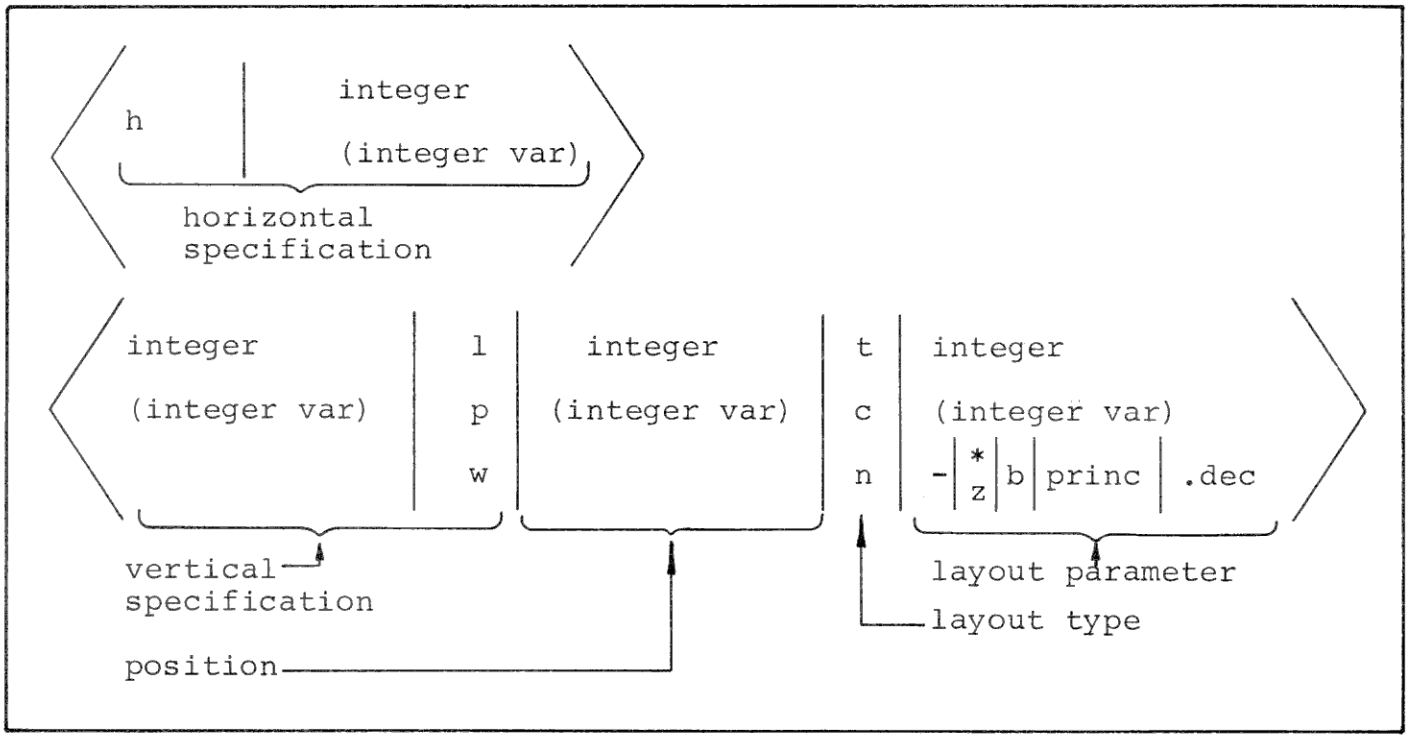


Figure 3.40: layout

See also the examples in figure 3.39 and 3.42.

3.2.12.2 Value Specification

If the layout contains a layout type (t, c or n), a value specification must be stated, whose type must correspond with the layout type.

text value

For a text layout (t) a text value must be stated, i.e.

1) a text constant (one or more text characters within apostrophes),
2) a text variable
3) a variable name, e.g. name (v32)

3.2.12.2 Value Specification

numerical value

For a numerical layout (n) or a character layout (c), a numerical value must be indicated; this may be:

4) a numerical constant
5) a character constant (1, 2 or 3 text characters between points)
6) a numerical variable
7) an expression of numerical variables and/or constants
8) a digit group extracted from a numerical variable by the operator, 'digits'
9) an integer expression stated with the operator, 'mod' or //.

The points 1-2 and 4-6 belongs to the concept 'value element' (cf. section 3.1.4). The expression in point 7 corresponds to what can be written after the assign symbol in a compute list (cf. section 3.2.3).

The digits-operators and the name-operator have been described under 'modify' (section 3.2.2).

The operators for the integer division and the modulo have also been described under 'modify'; note, however, that in these operations the layout must not contain decimals.

Automatic normalization

Numerical printing will always take place with an automatic regard to possible decimals in the value to be printed, so that the layout's decimal point always will follow right after the value's unit position, as shown in figure 3.41.

3.2.12.2 Value Specification

```
> print <1 1 n8>    : var_2_dec
        < 11 n5.2>  : var_2_dec
        < 21 n3.4>  : var_2_dec
        s

When var_2_dec contains the value 123.45, this
print operation will print the line:

  123   123.45     123.4500
```

Figure 3.41: processing decimals in print.

Between the layout and the value specification, a colon may be written (out of consideration for those who are used to write GENIUS programs) but this colon is not compulsory (out of consideration for those who are accustomed to use the previous Duet language).

Figure 3.42 shows a more extensive example of a print operation. The Duet compiler attends to the printing of the value specifications and the comments beneath one another, but it does not alter the structure of the layouts. We recommend that you provide a certain editing of the latter.

3.2.12.2 Value Specification

```
d10:  print    d11                                  ; possible head line
               <2l  4 t30  >  customer_name         ; text variable
               <  (v2)n5.2 >  balance - payed       ; new balance
               < l  4 t30  >  name(v32)             ; variable name
               <  (v2)nb5.2>  payed
               < h(v1)     >                        ; basic pos. for date
               d20                                  ; print date
               < h  0      >                        ; basic pos. zero
               < l 13 t10  >  customer_discount     ; fixed text
               <   23 c1   >  .:.                   ; colon
               <   25 n3.1 >  (v13(2) - v3*100/v3 ; discount
               <   30 c1   >  37                    ; p.c. character
               < l         >                        ; exhaust the last line
               s

d11:  modify   line_counter :+ 4
               s
   >  if       line_counter > max_line then d12

d12:  modify   page_no :+ 1
               line_counter := 0
               s
   >  print    <p 5 t30>      'list op customers and payments'
               < 50 t40>      'page'
               < 55 n3 >      page_no
               s

d20:  print    <  1 n2  >     digits (2,1) of date ; day
               <  3 n-z2>     digits (4,3) of date ; month
               <  6 n-z2>     digits (6,5) of date ; year
               <  3 c1  >     46                   ; point
               <  6 c1  >     46                   ; point
               s
```

Figure 3.42: print example

3.2.12.3 Printing with Standard Layout

In order to substitute the 'list' operation, which could be used for standard printing of variables in the old Duet language, two kinds of printing with standard layout have been introduced. They are both shown in figure 3.43.

```
print <l 1 t40> 'standard printing with var.names'
      <a 5,20>  numvar
      <a 5,20>  txtvar
      <l 1 t40> 'standard printing, fixed texts'
      <s 5,25>  'customer name', cust_name
      <s 5,30>  'balance', balance
      s
```

Figure 3.43: standard layout

The standard layout contains a layout type, 'a' or 's' and two position statements, integers separated by commas. Only a simple variable can be stated as a value specification.

The first position states the starting position for printing a variable designation. By layout type 'a' the variable's name is printed whereas the stated text constant is printed by the layout type 's'.

The specified variable is printed with a standard layout which takes the variable's type and possible number of decimals into consideration.

3.2.13 DB-operations

The Database-operations in Duet (db-operations) are based on the DBMS-procedures of the Soda-system (see ref. 2), as each db-operation in Duet calls the Soda-procedure with the same name. The format of the db-operations is as shown below in the examples in fig. 3.44.

```
> get          s1
> next         s (setno)
> lookup       s7
> put          s (v2)
> delete       s2
> newset       s7

> create       s3, i7
> create       s (v15), i (v16)
```

Figure 3.44: db-operations

The first parameter indicates the set no. For create furthermore, the record type must be specified. Both set no. and record type can be indicated with integers (e.g. s3 and i7), or by reference to a simple integer variable (e.g. 's(setno)' and 'i(v16)'. If the latter is the case the setno/record type is not decided upon till the start of the run.

The examination of the individual db-operations below is rather sketchy. For a close examination see the Soda-manual (ref. 2).

3.2.13 DB-operations

get

The operation 'get' fetches by direct access a record from the Database, and makes this record a 'current record' in the set. Values are transferred from record fields to variables according to the field associations in the ld-description containing '<'.

next

The operation 'next' fetches either the next record in the set after the one last read by 'next', or the first record in the set, when 'next' is called just after 'newset'. The record becomes 'current record' in the set. The value is transferred from record fields to variables by the same procedure as in 'get'.

lookup

The operation 'lookup' checks whether the record with the indicated keys is included in the set or not. Even though the record exists, is will not become a 'current record', and no field transfers are performed. 'Lookup' can only be used on sets belonging to cf-master files (set type M).

create

The operation 'create' creates a new record and makes this a 'current record' in the set. Variables attached to the current record type are initialized with a standard value according to the field associations in the ld-description containing '*'.

put

The operation 'put' delivers the 'current record' of the set to the Database. After 'put' no 'current record' exists.

3.2.13 DB-operations

By 'put' after 'get/next' an existing record in the Database is replaced by a new version, and values are transferred from variables to record fields according to the field associations containing '>' or '=>'.

By 'put' after 'create' a new record is inserted into the Database, and values are transferred from variables to record fields according to the field associations containing '->' or '=>'.

The location of the new record in the file depends on the set type as shown in the table, fig 3.45.

set type	file type	previous activity	record is inserted
M	cf-master	~	according to keys
B	bs-file	~	always at the end of file, (at eof)
L	cf-list,	~	insertion is not allowed
	cf-list subscripted	newset	as the first in the chain
		next	in front of prev. 'current record'
		end-of-chain	at the end of the chain

Figure 3.45: location of new records when inserted into the Database.

3.2.13 DB-operations

delete

With the operation 'delete' the 'current record' of the set is deleted from the Database. After this no 'current record' exists any longer in the set. No field transfers are performed. If the set belongs to a bs-file, all the succeeding records in the set, if any, are consequently deleted.

newset

Finally the operation 'newset' is designed to initialize a sequential scan of the file. No 'current record' exists in the set after 'newset', and no field transfers are performed. The starting position for a later sequential reading is defined by the set type as shown in fig 3.46.

set type	file type	position is defined
M	cf-master	according to keys
B	bs-file	identspec exists: according to keys no identspec: at the start of the file
L	cf-list, sing.	newset is not allowed
	cf-list, subscr.	at the front of the chain

Figure 3.46: position after 'newset'

result.soda

The result of the db-operation is accessible through the result variable defined as 'result.soda' with the following values:

```
result.soda  =  0: db-operation ok
             =  1: record does not exist
            >=  2: error in db-operation
```

3.2.13 DB-operations

Result 1 is accepted as a result to operations 'lookup' (the wanted record does not exist) and 'next' (end of chain/end of file), while this result after 'get' will cause the call of the error procedure 'duet_data_fejl'.

The result values 2 and forward always cause the call of an error procedure, either 'duet_data_fejl', 'duet_program_fejl', or 'duet_system_fejl' depending on the type of error (cf. section 5.3).

result.recno

After the insertion of a new record in a cf-list file or a bs-file the result variable result.recno contains the position of the new record. This position may later be used as a key for direct access to the file/set.

3.2.14 Select

The 'select' operation is a garbage action by which various run parameters can be altered dynamically during the execution of the Duet program.

The following items can be selected:

- print channel for result-, error - or test output,
- reaction on error situations,
- reaction on reading of standard mark, and
- set/clear test variables

The format of the select operation is a list of variable length terminated with a Duet stop (the character 's'). Each line in the list executes one of the above-mentioned actions, and the various actions can be mixed arbitrarily among each other as in figure 3.47.

```
> select   print  on  3
           test a :+ 0, 2, 23
           print test on 4
           return on data error
           exit p9 on system error 11, 12
           no stdassign on read
           s
```

Figure 3.47: example of a select list

In the following passage various possibilities are discussed in their respective subsections.

3.2.14.1 Select print

The print channel (i.e. subscript in the zone array 'prinz') for result output, error messages and test output, can be selected by 'select print' the format of these lines is shown in figure 3.48.

```
> select   print   on   v32
           print   data error on 6
           print   program error on 9
           print   system error on v16
           print   test on 0
           s
```

Figure 3.48: select print

channel no. The channel number can be stated with an integer in the interval 0-9, or with a simpel integer variable. The Duet interpreter checks that the channel number does not exceed the number of zones in 'prinz'.

out The channel number 0 means: print on the zone 'out'. This possibility, however, is only available for error messages and test output. The result output can only be printed on one of the 'prinz'-zones.

Standard values for channel numbers are used when the select print has not been called. Standard channel for result output is 1 (prinz (1)) and for error messages and test output 0 (out). These standard channels are selected each time the procedure 'init_duetmaskine' is called; the test channel, however, is only selected once when calling 'init_duet2'.

3.2.14.2 Select exit/return

The reaction after an error message is selected by 'select exit' and 'select return'. The standard reaction to any error is an exit from the Duet interpreter (cf. section 5.1). This can be changed selectively for each of the error types so that the Duet program either exits to a program point (cf. section 3.2.1.3) or returns in order to continue the program. The possibilities are shown in figure 3.49.

```
> select exit p3     on data error 2,7,5  ; exit to p3
        return on data error 6,8          ; proceed unchanged
        exit p0 on data error 11          ; exit out of program

        exit p0 on system error           ; exit for all system
                                          ; errors
        return on program error           ; proceed after all
                                          ; duet-program errors
        s
```

Figure 3.49: select error reactions

p0 cannot be defined as a program point. Here it is used to state an exit from the Duet interpreter.

If one or more integers are stated after the error type (data error, program error, system error) the reaction applies exclusively to the error number, specified that way. If the error number is omitted the reaction applies to all errors of the type in question.

3.2.14.3 Select stdassign

This defines whether, or not, the standard value is to be assigned to the read variable by reading the standard mark.

There are only two possibilities as shown in figure 3.50.

```
> select  stdassign on read    ; standard value
          s                    ; is assigned

> select  no stdassign on read; standard value
          s                    ; is not assigned
```

Figure 3.50: select stdassign

3.2.14.4 Select test

Some bits in the test variables: testa, testb, --- testg, testh can be set and cleared by means of the 'select test' as shown in figure 3.51.

```
> select  test a  :=  on
          test b  :=  off
          test c  :=  0,1,3,7
          test d  :+  3,4
          test e  :-  21,22,23
          test f  :=  duet_testvar
          test g  :+  v17
          test h  :-  v16
          s
```

Figure 3.51: select test

3.2.14.4 Select test

Note: a space is required between 'test' and the names of the test-variables ('a', 'b', etc.).

assign operator

The following assign operators can be adapted:

:= for ordinary assign
:+ for addition of further bits and
:- for deletion of single bits

test bits

After the assign operator follows a designation of the test bits which are to be set, added or deleted in the test variable. The test bits are numbered from the right with the numbers 0-23.

'On' or 'off' can be stated; they reset all testbits to 1 or to zero respectively. This possibility can only be used together with the assign operator :=.

Test bits may also be written as one or more integers separated by commas by which the bit numbers, of the test bits which are to be set/added/deleted, are stated.

Finally a simpel integer variable can be stated. Here the bit pattern, which is contained in this variable, is set/added/deleted in the test variable.

The test variables testa, testb, testc and testd are

The test bits

test d := 1, 2, 12

can be useful for the programmer, when debugging a new Duet program. The setting of these bits provides a dynamic trace or monitoring of the running program with the following test printing:

bit 1: perform <duetname>
bit 2: operation <opt.name>
bit 12: return <duetname>

3.2.15 Exit

```
> exit    p4    ; exit to the program point p4
```

Figure 3.52: exit

The format of the exit operation is shown in figure 3.52. It causes a return to the nearest, active program point with the same number as described in section 3.2.12.

The program point must be stated within the interval p1 - p9.

3.2.16 Algol

algol
special
action

The 'algol' operation is the user's escape possibility, by which Algol-coded special actions in the control program can be activated. This facility is used for program parts unfit for coding in the Duet language.

```
> algol 3 ('text constant' 12345,
           name(v27),
           adr(v27), adr(var(v16),
           customer_no).
> algol 15
```

Figure 3.53: algol

For the 'algol' operation an integer must be stated, indicating the number of the wanted special action.

Furthermore, a parenthesis can be stated, containing ~~an arbitrary number of~~ at most 7 parameters separated by commas and possible line-feeds. These parameters must be interpreted by the special action as described in section 5.4.

The permitted types of parameters are:

1) text constant
2) text variable
3) name of the variable specified by means of the 'name' operator
4) numerical constant
5) character constant
6) numerical variable
7) address of the variable specified by means of the 'adr' operator.

3.2.16 Algol

The 'name' operator has been described under modify (section 3.2.2.2).

adr ( )

The 'adr' operator states that the address of the variable, referred to, is requested. If the variable is an array, or if it is to be used as a return parameter (i.e. is assigned by the special action) it is necessary to use the 'adr' operator. It can be used as shown in figure 3.54.

```
> algol 2  (adr(simple_var),
            adr(array_var),
            adr(var (simple_var)),
            adr(var (array_var (subscr))) )
```

Figure 3.54: Algol action with adr-specified parameters

After 'adr(' a variable name (or -number) can be stated directly. A subscript must not be given to an array variable. The address which will be delivered to the special action will then be the field address of a specified, simpel variable, alternatively the field address of the first element of an array variable.

adr (var( ))

If 'adr (var(' is written instead, the specified variable must contain the variable number of the wanted variable. In that case it is the field address of this variable, stated indirectly, which is computed.

3.3 Maximum limits in duet operations

Execute

An execute list can contain 127-255 duet references, depending on the number of block references and program points appearing between the local duet references.

A line containing the declaration of a program point can contain no more than 7 duet references.

Modify Compute

A modify list and a computelist can contain at most 127 references to variables and/or constants.

Assign Case Action

An assign, case, or action list can contain 127 test lines.

Read

A read operation can contain up to 3 read specifications.

Print

In a print list no more than 127 layout elements can be specified. As layout elements are counted each of the following types:

duet	reference
p	(new page)
l	(line feed)
h	(horizontal specification)
n	(numerical layout)
t	(text layout)
c	(character layout)

A standard layout counts as 2 layout elements (a line feed and a numerical or text layout).

Select

The select operation can contain a maximum of 127 lines.

Algol

The algol operation can be specified with at most 7 parameters.

4.1 Program Text and Listing

text file

The Duet program text can be read either from a common text file on a disc or from a Sysdok file.

If the program text is stored in a Sysdok file, it is recommended to place the separate Duet blocks in co-ordinate subsections. By a partial compilation of a single block the subsection in question can be specified, and the time the compiler would use to skip the other Duet blocks could be saved. In this case, however, the Duet head can not be checked.

editing

When keying the program text it is not necessary to edit the lines as they appear in the examples in section 3. The compiler will take care of the editing itself, when the program is listed.

form feed

-+-

The listing will also be equipped with form feeds corresponding to the type of paper on which listing is being made. Apart from that, you can force a form feed in front of a Duet operation by writing the new-page symbol '-+-' and/or a stopcode character (ISO-value = 12). In front of a block-start only a stopcode character is allowed. A change of section in a Sysdok file will not cause a form feed, however.

external and internal lineno.

In the listing all lines are printed with two line numbers. The external line numbers are printed first i.e. Boss lineno. or Sysdok lineno. After that follows an internal lineno., which numbers the lines in a block from 1 and onwards/upwards. By error messages always the internal lineno. is being referenced.

When editing, the single lines are divided into columns as shown in fig. 4.1.

4.1 Program Text and Listing

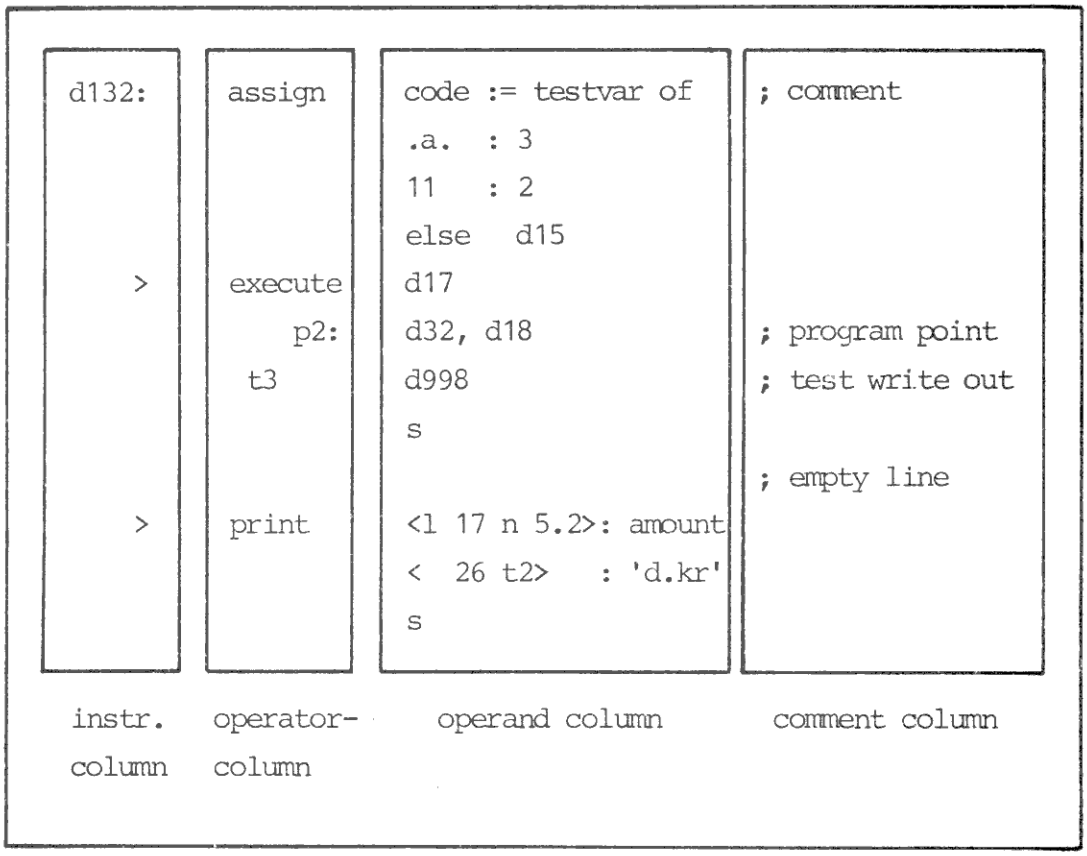


Figure 4.1 Line editing

The editing is performed so that line parts of the same type are always printed below one another. Within the operand - and the comment part the user must perform the editing, if any, (e.g. insert spaces to get the ':' placed under one another in an assign list).

In a printline, however, the compiler performs some editing as the print values here are edited below one another, but any editing within the layout must be controlled by the user.

4.1 Program Text and Listing

comment lines

Pure comment lines can be printed in two ways: If the comment character(;) is the very first in the line, the comment is justified left starting in the instructions column. If however there exist one space in front of the ';', the entire comment is printed in the comment column. This can be used to distinguish between a heading comment and ordinary comment lines.

4.2 Activating the Duetcompiler

In a run with the Duetabler, there are included 2 or 3 input files and a maximum of 4 output files, as shown in fig. 4.2.

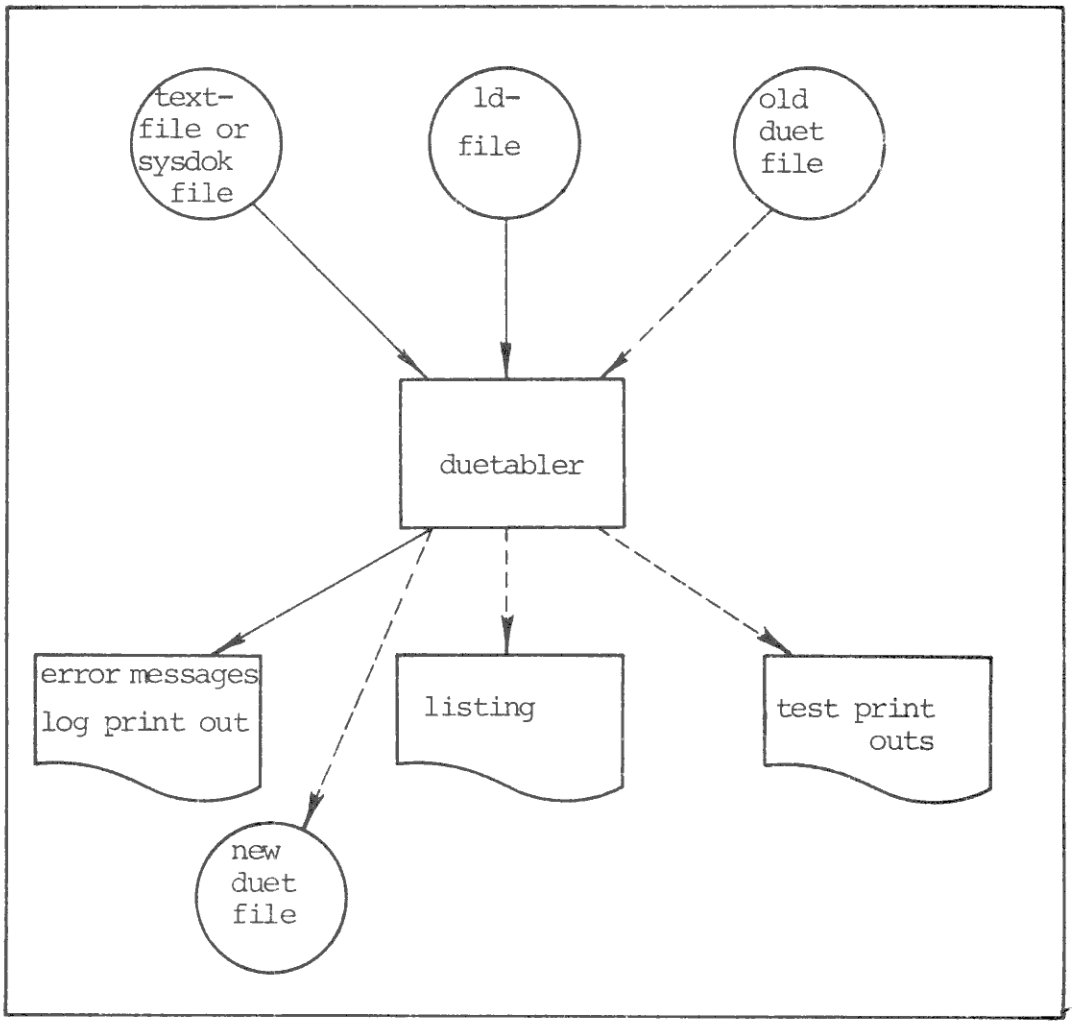


Figure 4.2: Survey of input for and output from Duetabler.

input

The input files are:

- a textfile or a Sysdok file containing the Duet program text,
- an ld-file (in the form of a descripfile) containing a compiled ld-description,
- perhaps an old Duet file from an earlier compilation of the Duet program.

4.2 Activating the Duetcompiler

output

The output files, of which only the first is always created, are:

- current output, where error messages, if any and a log printout, are written
- a disc area where the new Duet file is created,
- a disc area for a possible listing, and
- a disc area for test output, if any.

The use of these files are specified by fp-parameters when calling the compiler, as described below. The fp-parameters are mentioned in groups, where each group begins with a key-word followed by one or more parameters separated by points. Fig. 4.3 contains a survey of all parameter key-words.

4.2 Activating the Duetcompiler

```
parameters concerning source text:
        sysdok
        section
        vers
        duettext
        user
        init
        include

parameters concerning listing:
        list
        listout
        paper

parameters concerning ld files:
        descrip
        ldfile
        ldsection

parameters concerning duet files:
        oldduet
        newduet

parameters concerning compilation specification:
        insert
        change
        delete
        translate
        size

parameters concerning test print outs:
        test
        testout
```

Figure 4.3: outline of parameter key-words.

4.2 Activating the Duetcompiler

Most parameter groups can be left out, in which case standard values are used, as described for each parameter group. If a parameter type is mentioned more than once, the last will be valid except in case of the parameter types insert, change, delete, translate, and test, where further repetitions will supplement earlier appearances.

For examples of calling the compiler see section 4.3.

Below is a description of each parameter group.

Parameters concerning source text:

Sysdok file

The source text for the Duet program can be read either from a Sysdok file or from an ordinary text file. When read from a Sysdok file the following parameter groups should be mentioned:

```
sysdok.<sysdokreg_name>
section.<section_number>
vers.<version_number>
```

Sysdok

The Sysdok parameter states which Sysdok file the text is to be read from.
Standard: sysdok.sysdokfile

section no.

The section parameter states which section in the Sysdok file the text is to be read from. The section number is given in the usual Sysdokmanner with the main section and subsection numbers, if any, separated by decimal points. The parameter cannot be left out, if the text is read from a Sysdok file.

4.2 Activating the Duetcompiler

version — 'Vers' states, the version of the Sysdok section to be translated.
Standard: last version

textfile — When reading from an ordinary text file, the name of the text area, from where the Duet program is to be read, is stated with:

duettext

```
duettext.<duettext_name>
```

Standard: If this parameter is left out, the compiler reads from the Sysdok file. If the parameter is stated, no Sysdok parameters must then be stated.

The parameters below may be stated whether, or not there is read from Sysdok or a text file:

user number

```
user.<user number>
```

indicates the user number of the compilation. Only Duet blocks belonging to one user can be compiled in a single run, and the ld description used must belong to the same user or to a common user (user no = 0).
Standard: user.0.

11 karaktere?

initials

```
init.<initials>
```

states the initials of the programmer, who has activated the Duet compilation to be printed in the Duet log. Standard value (= empty teststring) should not be used.

4.2 Activating the Duetcompiler

include

include.<suppress_limit>

The suppress_limit is an integer in the interval 0-9. This parameter denotes suppression of those test-lines in an execute-list, which are equipped with a test-number larger than the suppression limit.
Standard: include.0, i.e. no test lines are included.

Parameters concerning listing:

list. { yes / no }
listout.<listout_name>
paper. { <boss_paper_format> / <lines_per_page>.<characters_per_line> }

list

The list parameter states whether a listing of the Duet program is created or not. (Possible error messages are in either case printed on 'current out', as well).
Standard: list.no

listout

The listout parameter states the name of the disc-area on which an edited listing is to be written. If the area does not exist already, it will be created as a temporary file, which is automatically converted on the local printer. If the area exists already, the user himself must provide for the converting. ~~This parameter will automatically set the parameter list.yes.~~
~~Standard: listout.listout.~~
Standard: listout.listout

4.2 Activating the Duetcompiler

paper

The paper parameter indicates, how the edited Duet program is to be printed. A single integer can be stated, giving the Boss paper format, or 2 integers which decide the number of lines per page and the number of characters per line.

In the edited listing form feeds are printed:

1. when <lines_per_page> are exeeeded
2. before each block start
3. by new-page symbol ('-+-' or stop code in input

<boss_paper_format>:
0: monitor paper (<lines_per_page> = 62)
2: A4-horizontal (<lines_per_page> = 40)

(A4-vertical = 1 is not allowed)

<lines_per_page>.<characters_per_line>:
These numbers can be stated explicitly instead of the Boss paper format. However, <character_per_line> is not used for the moment. If <lines_per_page> are less than 15, form feed rule no. 1 is cancelled, which results in a more compact listing.
Standard: when reading from the Sysdok file: as stated in the Sysdok file's owner information.
When reading from text file: paper.0.

4.2 Activating the Duetcompiler

Parameters concerning the ld-file:

ld file

```
{descrip}                                    {.<ld_version_no>}1
{ldfile}  .<ld_file_name>                    {            }0

ldsection.<ld_section_no>
```

indicates the name and section of the description file, where the compiled ld-description is stored. The words 'descrip' and 'ldfile' can be used at pleasure. The parameter 'ldsection' cannot be left out.
Standard for ld-filename: descrip.descripfile

Parameters concerning the Duet file:

old- and new Duetfile

```
oldduet.<old_duet_file_name>
newduet.<new_duet_file_n
```

Add the following description of ld_version_no:
If the ld_version_no is specified different from zero, the compiler will check that the version of the ldfile matches the specified version. If ld_version_no is zero or is left out, the version check will be suppressed.

indicates the names ...ie, respectively. If ... ...ied, this version number ... with the version numb...

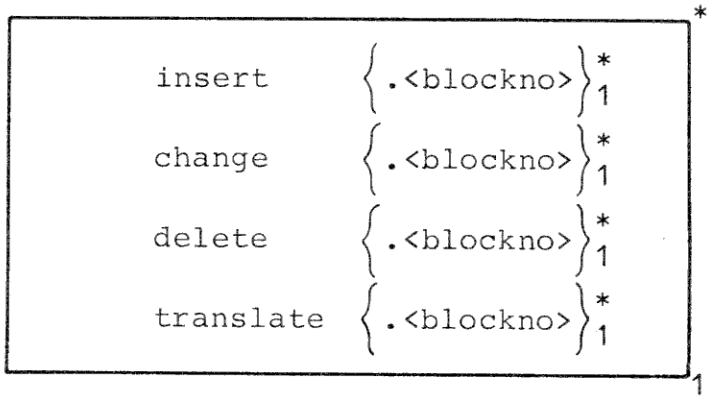
If the 'newduet ...ft out, the 'oldduet' is not allowed eit... Duet file is created. The compilation run ...en become only a check of Duet blocks.

If newduet is stated, but not oldduet, a new Duet file is created with version number 1, consisting of correctly compiled Duet blocks.

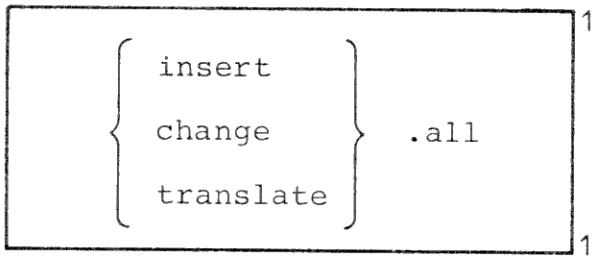
4.2 Activating the Duetcompiler

If both newduet and oldduet are stated, blocks from the old Duet file are merged with the correctly compiled blocks.

Parameters concerning the compilation specification:



or



These parameters, which must be stated after the Duet file parameters, indicate which Duet blocks are to be read by the compiler. If 'insert.all', 'change.all' or 'translate.all' are state, all Duet blocks found in the source text are compiled.

If no compilation specifications are stated, the compiler will assume "translate.all".

4.2 Activating the Duetcompiler

insert — The specified blocks are inserted in the Duet file. This is only allowed if 'newduet' is stated. If 'oldduet' is also stated, the specified blocks must not be found in the old Duet file.

change — The specified blocks are replaced by freshly compiled Duet blocks. This is only allowed if both 'oldduet' and 'newduet' are stated.

delete — The specified blocks are deleted from the Duet file. This is only allowed if both 'oldduet' and 'newduet' are stated.

translate — The specified blocks are checked by the compiler, but are not inserted in the Duet file. Is always allowed.

block no. — A block number can only appear once. Blocks found in an old Duet file, if any, but not mentioned in the list, are transferred unchanged to the new Duet file. The same holds for blocks specified under translate, and for incorrect blocks.

size

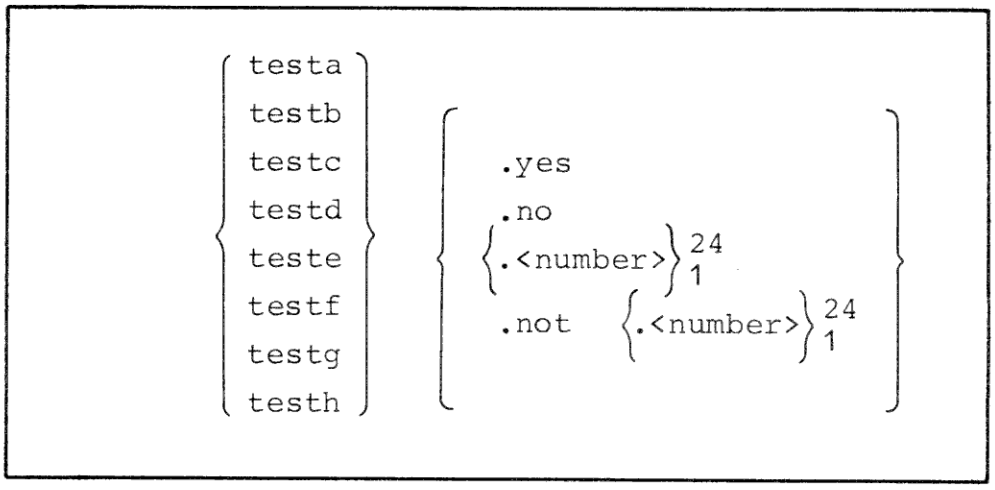
size.<extension_per_cent>

This parameter states in percentages the expansion of all the compiler's internal tables. It should only be used, if the compilation terminates with an index-error; a new run can be attempted using the size-parameter. The maintenance group must be notified, when the use of the size parameter becomes necessary. Size 100 means a doubling of all tables. Standard: size.0.

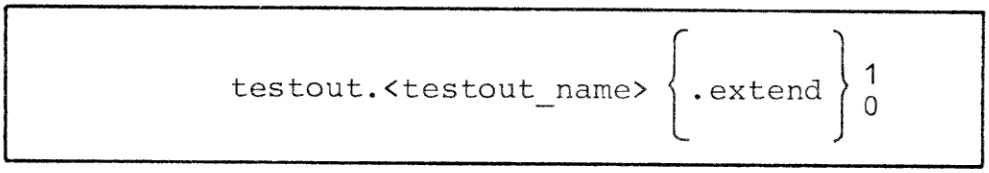
4.2 Activating the Duetcompiler

Parameters concerning test output:

These parameters must only be used in agreement will the maintenance group, when an error in the Duet compiler occurs.



Indicates which testbits are to be inserted in the respective test variables. Testbits are numbered from 0 to 23.
Standard: no textbits



Indicates the name of the area, where the test output is printed. If '.extend' is stated, the area is extended, if necessary, to contain the test output. If not, the test output is written cyclically in the area. The testout area is neither created nor converted automatically.
Standard: testout.testout

4.2 Activating the Duetcompiler

If syntactical errors are discovered in a parameter group, an error indication is printed in the form of:

<*< if a new parameter group appears as an illegal termination of the previous parameter group, or

<*> if an error is found in a keyword or in a parameter in the parameter group in question.

After errors in the fp-parameters, whether they are syntactical errors or errors in consistency checks, the run is terminated.

4.3 Resource Demands

The resource demands for the compilation job vary with the size of the Duet program, and the figures below should therefore only be regarded as a guide. Only Boss parameters exceeding the standard are included.

size 80000	In 100000 bytes no extra segment transports take place. Minimum is 60000.
area 8	can be reduced, if some of the output files are not created.
time	In 80000 bytes it takes app. 1 minute to compile a maximum size Duet block and app. 20 sec. to skip one. In 60000 bytes the corresponding times are 7 and 3 minutes.
perm	A Duet block of maximum size takes up the space of 5-8 segments in the Duet file. If this already exists additional space for an expansion, if any, must be taken into account.

4.3 Resource Demands

Examples of jobfiles.

```
job eah 28xxxx size 80000  area 8  time  3 0,
    perm disc 100 1
mode list.yes

duetfile = set 1 disc                    ; create new duet file
scope user duetfile

duetabler,
   duettext.eahduet,                     ; source text
   init.eah,
   user.0,
   ldfile.eahdescrip ldsection.30,       ; ld-file
   list.yes paper.2,                     ; listing
   newduet.duetfile,                     ; new duetfile
   insert.all,                           ; the whole duet file

finis
```

Figure 4.4 Create a new Duet file

The example in fig. 4.4 shows the creation of a new Duet file in 'duetfile' from a program text in 'eahduet'. The listing on A-4 horizontal is created in 'listout', which is automatically converted on the local printer.

4.3 Resource Demands

```
job pl 28xxxx  size 80000  area 8   time 3 0
mode list.yes

oldduetfile = move duetfile
listout = copy 0                          ; create list out area

duetabler,
   sysdok.eahsysfile section.22.1  ; source text
   init.pl,
   user.0,
   ldfile.eahdescrip ldsection.30, ; ld-file
   list.yes                        ; listing
   oldduet.oldduetfile,            ; old duet file
   newduet.duetfile,               ; new duet file
   change 1.7,
   delete.2

convert listout std
finis
```

Figure 4.5: change the Duet file

Fig. 4.5 shows the correction of a Duet file. The Duet blocks 1 and 7 are re-compiled, while block 2 is deleted. The other blocks remain unchanged. In order to keep the newest version of the Duet file in the same area, 'duetfile' is moved to 'oldduetfile' before the compilation.

The Duet program text is read from section 22.1 in the Sysdok file 'eahsysfile'.

The listing is to be printed on the central printer, hence the listout area is created before the compilation.

4.4 Log Print Out

As a conclusion of a Duet compilation, where a new Duet file is created, a Duet log is written, showing the result of the run and the contents of the old and the new Duet file. An example is shown in fig. 4.6.

The 'blocksize' contains two numbers. The first one states how many words the compiled block takes up. This number must not exceed 2047.

The other number states the length of the block including numerical constants and text constants. This figure is of importance to the dimensioning of the Duet array in the control program (cf. section 3.1.2 and 5.2.1).

The rest of the information is selfexplanatory.

4.4 Log Print Out

```
duetabler log       duetblocktest                eah.7   7.03.1977 - 16.03

binary files:   area            version date         ident

new duetfile:   newduetfile     7 eah   070377.1603  31    duetblocktest
old duetfile:   eahduetfile     6 pl    030377.1619
localdata   :   eahdescrip      0 eah   070377.1320  30    variableId

textfile    :                       section

sysdok      :   eahsysfile     24     22.      duetblocktest

    duetabler
block command   old    ! block user duet-   ld-     date        blocksize
  no          version  !   no       vers.   vers.               duet/total
   1  insert          !  1+   0    7 eah    0    070277.1603     16/23
   2  copy      6 pl  !  2    0    6 pl     0    030377.1619     22/29
   3  change    6 pl  !  3+   0    7 eah    0    070377.1603      9/16
   4  transl    4 isc !  4    0    4 isc    0    030377.1400      6/14
   5  insert          !  5+   0    7 eah    0    070377.1603      6/14
   6 -insert          !
   7 -change          !
   8 -delete          !
   9 -transl          !
  10  insert          ! 10+   0    7 eah    0    070377.1603    144/219
  11  insert          ! 11+   0    7 eah    0    070377.1603     80/98
  12 -insert          !
  13 -change          !
  14 -transl          !
  15 -delete          !
                        total used 8 segments of 36

- : error: no change/insert/delete
+ : block  inserted or changed
```

Figure 4.6: Duet log

4.5 Error Messages

All error messages from the Duet compiler are printed on 'current output', as well as in a possible program listing.

In the listing the error message will have the form shown below:

***** <error no> : <error text> { <error parameter> }$_{0}^{3}$

and the incorrect line will be printed unedited, beginning with **.

In 'current output' the error will be printed as follows:

<lineno><charno><text> <error no> : <error text> { <error parameter> }$_{0}^{3}$

the incorrect line will then be written out. <text> is the syntactical unit, causing the error, and <charno> states where in the line this unit begins. <lineno> is the interval line number.

At this point before each error is described in error numerical order, we shall state some general remarks.

After an error the compiler will try to continue the compilation, as soon as possible. By many errors, however, it is necessary to skip the rest of the line. This is stated in the explanation to each error.

4.5 Error Messages

In some error situations it is necessary, however, to skip until the beginning of the next Duet operation. In that case the succeeding lines will be cancelled and labelled error type 1: SYNTAX, even though they might not be incorrect.

If an error is discovered during the compilation of a Duet block, the latter is not delivered to a Duet file, if any, and this block will be marked in a special way on the Duet log (cf. section 4.4).

Errors in one block will not affect the compilation of the other blocks. Compilation time can therefore be saved, if only the altered blocks are recompiled, after the error correction.

1: SYNTAX: <text>

<text> is the last read syntactical unit.
This syntactical construction is not allowed.
The rest of the line is skipped.

2: ILLEGAL VARIABLENO <v_no>

A variableno. must not exceed the largest variable number used in the ld-description. In order to continue the compilation, a reference is simulated to a simple word var without decimals.

4.5 Error Messages

3: UNDECLARED VARIABLE <text>

The last read syntactical unit is an identifier or a variable no., which does not correspond to a variable declared in the ld-description. In order to continue the compilation, a reference is simulated to a simple word var without decimals.

4: ILLEGAL ERROR NUMBER <error_no>

System error. Contact the maintenance group.

5: ILLEGAL OPERAND VALUE <value>

Is used where: suppression specification > 9
Duet program point > 9
select print channel > 9
select ... on error: illegal error no.
select test: testbit > 23
select test: testbit double specified
number <> 0 in 'modify array := 0'
number <> 0 in 'modify recno := 0'
Algol action no < 1

The rest of the line is skipped.

6: NOT SIMPLE VARIABLE <var_name><var_no>

Neither array variable nor array element are allowed here.

4.5 Error Messages

```
7:  ILLEGAL VARIABLE TYPE <var_name> <var_no> <var_type>
```

This variable type must not be used in the given context.

```
8:  ILLEGAL NUMBER <no>
```

A numerical constant is stated outside the range of a 24-bits word. The error appears in connection with test values/assign values in an assign- or action list.

```
9:  UNDEFINED DUETNAME <d_name>
```

The Duet name referred, is not declared in any Duet instruction within the block.

```
10:  ILLEGAL NUMBER OF DECIMALS <var_name> <var_no>
```

The variable has not been declared with decimals and can therefore not be used:

as subscript
for reading in of a character field
as a test variable in assign,
action, case, and for

The rest of the line is skipped.

4.5 Error Messages

11: ILLEGAL BLOCKNO <block_no>

The block no must not exceed 255.
If the error appears in a block head, the rest of the program text is skipped.
By error in a block reference the compilation is continued.

12: ILLEGAL USERNO <user_no>

The user no. must not exceed 127.

13: USERNO INCOMPATIBLE WITH LD DESCRIPTION <user_no> <ld_user>

<user_no> userno. in block head
<ld_user> userno. in ld-description
The user no. in block head and ld-description must be the same, unless one of the numbers are 0.

14: ILLEGAL ENTRYNO <entry_no>

An entry number must not be defined outside the interval 1-63, and not be referenced outside the interval 0-63.

4.5 Error Messages

```
15:  ENTRYPOINT PREVIOUSLY DEFINED IN LINE <line_no>
```

<line_no> the linenumber, where the entrypoint is first defined.

```
16:  ILLEGAL END NUMBER <block_no> <end_no>
```

<block_no> from the block head
<end_no> from the end-line

The two numbers must be identical

```
17:  NB! BLOCKSIZE APPROACHING LIMIT <size> <max_size>
```

<size> the length in words of the compiled Duet block.
<max_size> the maximum block length in words.
The space available for possible block extensions is limited.
This is only a note, which does not affect the creation of a Duet file.

```
18:  DUETNAME PREVIOUSLY DEFINED IN LINE <line_no>
```

<line_no> the line number, where the Duet name is defined for the first time.

19: not used.

4.5 Error Messages

20: ILLEGAL NUMBER OF CHARACTERS <no>

With a read specification of type c, a maximum of three characters can be read.
The rest of the line is skipped.

21: ILLEGAL TERMINATOR CODE <code_value>

Only the terminator codes 0, 1, 2, 3, are alowed in a read specification.
The rest of the line is skipped.

22: TOO MANY READSPEC

No more than 3 read specifications are allowed in a 'read' specification.
The rest of the line is skipped.

23: ILLEGAL CASENO <case_no> <no>

<case_no> the read number
<no> the expected number
There is a discrepancy between the number of Duet actions in the case list and a specified case number.
The rest of the line is skipped.

4.5 Error Messages

24: TOO MANY DUETREF IN ONE LINE

In an execute list no more than 7 Duet names are allowed in a line which has a Duetpoint attached to it. If a Duet point is to include more than 7, Duet actions, these must be specified in a separate execute list.
The rest of the line is skipped.

25: ILLEGAL USE OF D0

d0 is not allowed in: the entry point definition,
the execute list,
the if-operation after 'else'.
The rest of the line is skipped.

26: NB! DUETBLOCK TOO BIG <size> <max_size>

<size> length in words of the compiled Duet block.
<max_size> maximum block length in words.
The block must be split up into two smaller blocks.

27: ILLEGAL DIGITS SPEC <no_a> <no_b>

In 'digits (a,b) of' it is required that a>=b.

4.5 Error Messages

28: ILLEGAL SUBSCRIPT VALUE <subscr_values>

A constant subscript value has been specified outside the limits of the array variable in question.
The rest of the line is skipped.

29: ILLEGAL SETNO <set_no>

The Duet compiler does not check that a set is defined in the ld-description, but only that the set number is within the interval 1-max_def_setno.
The rest of the line is skipped.

30: ILLEGAL RECORDTYPE <no>

The record type is not allowed.
The rest of the line is skipped.

31: ILLEGAL CHARACTER <text>

the last read syntactical unit contains an illegal character.
The rest of the line is skipped.

4.5 Error Messages

32: ILLEGAL BYTE <text>

the last read syntactical unit cannot be recognized.

33: ILLEGAL DUETNAME <d_name>

the Duet name must not exceed d1023.
The rest of the line is skipped.

34: TOO MANY DIGITS IN NUMBER <no>

the number of digits in a number must not exceed 15.
The rest of the line is skipped.

35: ILLEGAL IDENT <text>

the last syntactical unit read is identified as a variable identifier, but the variable is not known.
The rest of the line is skipped.

36: not used.

4.5 Error Messages

37: LISTAREA CANNOT BE CREATED <area_name> <no>

<area_name> name of the listarea
<no> the error value from the creation attempt.
Resources (segments, catalogue entries, or area-processes) for the creation of an area for listing are not available. The Duet program is compiled without listing.

38: BLOCKEND MISSING <block_no>

<block_no> the Duet block, last compiled.
The Duet program text is terminated (end_medium/ end_of_section) without a block end.

39: DOUBLE USED VARIABLE NAME <var_name> <var_no>

Two variables in the ld-description declared with the same name. System error. This error should have been caught by the ld-compiler Contact the maintenance group.

40: ASSIGN SYMBOL MISSING

:= is missing in a line of a compute list.

4.5 Error Messages

41: ILLEGAL DELIMITER <text>

<text> the illegal syntactical unit.
Syntactical error in a numerical expression, the rest of the line is skipped.

42: ILLEGAL OPERATOR <text>

<test> the illegal syntactical unit.
Syntactical error in a numerical expression.
The rest of the line is skipped.

43: ILLEGAL OPERAND <text>

<text> the illegal syntactical unit.
Syntactical error in a numerical expression, the rest of the line is skipped.

44: 'OPTSTAK' OVERFLOW <no>

Too many bracket levels in a numerical expression.
The rest of the line is skipped.
The expression should be split up, and the intermediate results explicitly stored.

4.5 Error Messages

45: 'OPDSTAK' OVERFLOW <no>

see 44.

46: WORKVAR OVERFLOW

A numerical expression is too complicated to be computed with the 4 existing working registers. The rest of the line is skipped.

47: ILLEGAL OPERATOR COMBINATION

Syntactical error in numerical expression, the rest of the line is skipped.

48: ILLEGAL LENGTH OF TEXT/AGGR <length1> <length2>

By 'modify array1 := array2' it is required that the length of a single element in both arrays is the same.

<length1>, <length2> } the number of text characters/aggregat bytes per element in the two arrays.

By 'modify' of simple text-/bits variables it is required that the length of the left side variable is >= the length of the right side variable.
The rest of the line is skipped.

4.5 Error Messages

49: SYNTAX <text>

<text> the last read syntactical unit.
This syntactical construction is not allowed.
The compiler goes on to read the next syntactical unit in the line.

50: not used.

51: NOT DUETFILE IN OLDDUETFILE

The area specified as 'oldduet' does not contain any Duet file. No new Duet file is created.

52: ILLEGAL VERSION OF OLDDUET <version> <old_version>

<version> specified as an fp-parameter.
<old_version> version of the old Duet file.
If the Duet version is specified as an fp-parameter, the two version numbers must be identical.

53: not used.

54: DUETWORK CANNOT BE CREATED

Resources (segments, catalogue entries, or area-processes) for the creation of a working area for compiling Duet block area not available.
No new file is created.

4.5 Error Messages

55: MISSING SUBSCRIPT <var_name> <var_no>

The variable specified is an array variable, the subscript must be stated. The rest of the line is skipped.

56: ILLEGAL VERTICAL OPERAND OR POSITION

In a print lay-out no more than 70 line feeds are allowed at a time (corresponding to one page). The print position must be stated in the interval 1-127.

57: ILLEGAL NUMBER OF PRINCIPALS OR DECIMALS

In a print lay-out no more than 15 principals and 7 decimals are allowed. If the decimals are stated, neither the principals nor the decimals must be 0.

58: POSITION + FIELD LENGTH > 132

The field cannot be printed within the given line buffer of 132 characters.

59: BLOCK LOCATION ERROR

System error by updating of 'newduet'.
Contact the maintenance group.

4.5 Error Messages

60: BLOCK SEQUENCENO ERROR

Block numbers in the text must appear in an ascending sequence. The block is skipped.

61: SYNTAX OR ILLEGAL VARTYPE <text>

<text> is a variable no. or - name.
Is either a variable stated, where it is not allowed, or the variable type in question is illegal.
The rest of the line is skipped.

62: SYSTEM ERROR: VARIABLE NAME ADDRESS CONFLICT <no> <adr>

System error during the creation of tables for recognizing variable names. Contact the maintenance group.

63: SYNTAX ERROR BEFORE BLOCKSTART <text>

<text> is the last read syntactical unit.
Before block start only empty lines and comment lines are allowed. The rest of the text up to the next block start (begin <block_no>:) is skipped without any further print out.

4.5 Error Messages

64: UNKNOWN DUETBLOCK SPECIFIED FOR TRANSLATION <block_no>

The specified Duet block does not exist in the Duet program text.

65: SYSTEM ERROR: MISSING GOTO IN BYTEACTION <no>

A programming error in the Duetabler-compiler, contact the maintenance group.

66: INSERT DUETBLOCK: EXISTING BLOCKNO <block_no>

The specified Duet block is compiled (syntax checked), but is not inserted in 'newduet'.

67: ILLEGAL USERNO FOR CHANGE/DELETE DUETBLOCK <block_no>

The specified Duet block belongs to another user. In case of 'change' the block is compiled (syntax checked) but is not delivered to 'newduet'.

68: UNKNOWN DUETBLOCK SPECIFIED FOR DELETE <block_no>

The specified Duet block does not exist in 'oldduet'.

4.5 Error Messages

69: TOO MANY MOVEWORDS IN MODIFY ARRAY <no>

The total number of words to be moved in a single modify line must not exceed 2047.
The array must either be moved one element at a time, or it must be split up in shorter arrays in the ld-description.
The rest of the line is skipped.

70: DUET PROGRAM HEAD MISSING

When compiling to create a new Duet file the program head must be read too.

71: PROGRAM NO IN TEXT DOES NOT MATCH WITH OLDDUET <program_no> <old_proram_no>

<program_no> program no. read in Duet program text.
<old_program_no> program no. in old Duet file.
The two numbers must be identical.

72: DUET PROGRAM NAME TOO LONG

The Duet program name must not contain more than 17 characters. The name is cut off to 17 characters.

4.5 Error Messages

73: SPECIFIED LD-NUMBER DOES NOT MATCH LD-FILE <ld_no> <old_ld_no>

<ld_no> ld-number read in the Duet program text.
<old_ld_no> ld-number stored in 'oldduet'.
The two numbers must be identical.

74: SYNTAX ERROR IN DUET PROGRAMHEAD

The compilation is terminated.

75: ILLEGAL LEFT SIDE VARIABLE <var_name> <var_no>

By modify with modulo or an integer division the left side variable must be a numerical integer variable declared without decimals.
The rest of the line is skipped.

5. The Duet System in a Control Program

A compiled Duet program, stored in a Duet file, can be executed by a control program, containing the interpreter the 'Duet interpreter'.

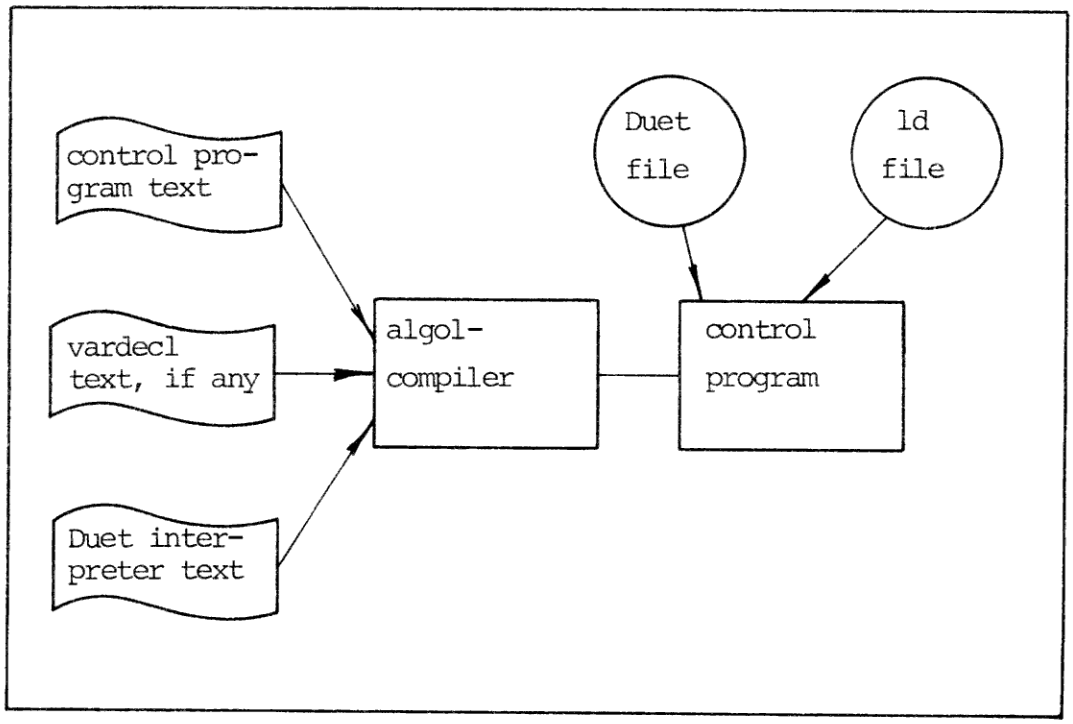


figure 5.1: The executing Duet system.

This section describes how a control program is established.

The 'teleop' from the Teledata system and the 'telescop' from Data Entry system are examples of such control programs. Therefore readers, who are going to code Duet programs for these systems, only have to make a superficial reading of this chapter.

5.1 Duet Texts and Algol Block Structure

The Duet interpreter consists of three texts which in suitable places must be incorporated into the control program with the 'algol copy' as shown in figure 5.2. Furthermore, a potential declaration file, generated by the Soda-ld compiler, must be incorporated.

5.1 Duet Texts and Algol Block Structure

```
begin   <*external block of the control program*>
   .
   .        declarations in the external block
   .
   algol copy.duettext1;          <*duet declarations*>
   .
   .        initialising in the external block
   .
   init_duet1;                    <*duet initialising*>

   begin  <*internal block of the control program*>
   .
   .        declarations in the internal block
   .
   procedure duet_algol(no);      <*must be declared by the user*>
   algol copy.duettext2;          <*duet declarations*>
   algol copy.vardecl;            <*variable decl. from soda-ld*>
   .
   .        initialising in the internal block
   .
   init_duet2;                    <*duet initialising*>
   .
   .        control program before
   .        the duet interpreter
   .
   algol copy.duettext3;          <*executing duet interpreter*>
   .
   .        user's termination
   .
   close_duet;                    <*close the files of the data base*>
  end internal block;
end  external block;
```

figure 5.2: The structure of a control program

5.1 Duet Texts and Algol Block Structure

The three texts are called:

duettext1
duettext2
duettext3

These texts incorporate the dbms-procedures of the 'soda' system in the form of 5 texts

sodatext1
sodatext2
sodatext3
tflytproc
ldfields

and the Duet interpreter calls the external procedure

read_general

and the 'primula' procedures, which can be found in textform in

prim

These files must all be present on disc, when the control program is compiled. The 'primula' procedures are compiled by the call 'i prim' before the call of the Algol compiler.

duettext1 'duettext1' contains partly declarations of variables, which are used as an upper limit for the array declarations in the 'duettext2' and partly the initializing procedure 'init_duet1' and the error printing procedure 'print_data_error'.

5.1 Duet Texts and Algol Block Structure

'print_data_error' is declared in block1 in order to give the user the possibility of changing the wording of the data error messages by declaring a similar procedure in an internal block.

Finally 'duettext1' incorporates the texts 'sodatext1' and 'ldfields' from the Soda system.

duettext2 — 'duettext2' contains a declaration of all the remaining variables and procedures of the Duet system. Furthermore it incorporates 'sodatext2', which incorporates both 'sodatext3' and 'tflytproc' from the Soda system.

duettext3 — 'duettext3' contains the actual interpreter, i.e. the executing part of the Duet interpreter.

5.2 Initialisation and Termination

The Duet interpreter is initalised by calling the initialising procedures 'init_duet1' and 'init_duet2'. If the control program is build up as a loop which activates the Duet interpreter several times, the 'init_duetmaskine' procedure must also be called before each new activation. (The first time, is 'init_duetmaskine' called from 'init_duet2').

After the last activation of the Duet interpreter the control program must terminate the DBMS and the 'prinz' zones by calling the 'close_duet' procedure.

5.2.1 Integer Procedure 'init_duet1(z)'.

The 'init_duet1' procedure reads the control records in from the ld- and Duet files and this causes an initialisation of a number of array limits.

duetlog (z) Furthermore a Duet log is written on the zone, which is specified as parameter for the call. It can either be the zone 'out' or 'fejlud'. The latter must be opened by the control program.

Before the call of 'init_duet1' the control program must have initialised the following variables:

duet_areal	the number of words which are reserved in the Duet array for placing active blocks.
duet_reg_navn	the name of the Duet file.
duet_version ld_version	if these are different from 0 they must be equal to the actual versions of the Duet- and ld file.
ld_reg_navn ld_afsnit_nummer	name- and section number of the descripfile, containing the ld file.
max_kanal	the number of zones in the zone array 'prinz'.
sd_extend_buf	the size of the buffer extention reserve, requested per cf-master-file (for possible use of extend_cf).

5.2.1 Integer Procedure 'init_duet1(z)'.

The return value of the procedure is a bit pattern which states the result of the call. If all bits are zero the result is ok. The single bits have the following significance:

```
1          : no ld file in 'ld_reg_navn'
1 shift 1  : the ld file is not formed by Soda-ld
1 shift 2  : the ld description is not correctly compiled
1 shift 3  : illegal version of the ld file
1 shift 4  : the ld-control record cannot be found

1 shift 12: check sum error in the Duet control record
1 shift 13: no Duet file in 'duet_reg_navn'
1 shift 14: not used
1 shift 15: illegal version of the Duet file
1 shift 16: the Duet area is too small for the largest
            Duet block
```

5.2.2 Integer procedure 'init_duet2';

The 'init_duet2' procedure reads in the rest of the ld- and Duet files, resets all variables, declared in the ld-description, to zero and initialises the character set table for the use of the read operation. Furthermore the 'prinz' zones are initialised for possible 'primula' print, and the 'init_duetmaskine' is called (cf. section 5.2.3).

Before calling the 'init_duet2' the user program must have executed the following initialisations:

prinz — all zones in the zone array must be opened to disc areas or tape files, but the zones must not be used for output until after 'init_duet2'.

readz — must be opened to the input area, if input is to be read.

After calling the 'init_duet2', the following variables can be redefined by the control program, if necessary:

alfa — the character set table can be changed, after which the 'intable (alfa)' must be called once more (cf. 'read', section 3.2.11 and section 5.5.1)

max_print_pos — (=132) the length of the line buffer for print

max_ref — (=20) see section 5.5.1.

test_kanal — (=0) test output is printed as standard on the zone 'out', but with the 'test_kanal' it can be diverted to one of the zones in 'prinz'.

5.2.2 Integer procedure 'init_duet2';

Furthermore the variables, assigned by the 'init_ duetmaskine', can be redefined, see the next section.

The return value of the procedure is a bit pattern which indicates the result of the call. If all bits are zero the result is ok. The value '1 shift 3' indicates that the Duet file does not contain any blocks, correctly compiled. All other error bits are due to system errors. If such errors appear, please contact the maintenance group.

add.

The value '1 shift 23 + 40' indicates that a bs-file specified in the ld-description does not, exist at runtime.
The value '1 shift 23 + 41' indicates a block-length-error in a bs-file. (If a cf-masterfile or a cf-listfile referred in the ld-description is missing at runtime, the program is terminated by an alarm message).

5.2.3 Procedure 'init_duetmaskine';

The 'init_duetmaskine' procedure puts the Duet interpreter into a neutral state. It is called from the 'init_duet2' procedure before the first activation in the Duet interpreter and can be called from the control program before subsequent activations, if any.

It initialises the following variables which afterwards can be redefined by the control program as required:

```
kanal   :=   1;
```
'prinz' zone no. for result print outs.

```
data_fejl_kanal      := 0;
program_fejl_kanal   := 0;
system_fejl_kanal    := 0;
```
error messages are printed as standard on the zone 'fejlud', but by changing the channel number it can be diverted to one of the zones in 'prinz' (cf. 'select', section 3.2.14).

```
data_fejl_akt        (1: max_data_fejl)
program_fejl_akt     (1: max_programfejl)
system_fejl_akt      (1: max_systemfejl)
```
are all reset to zero, signifying: exit from the Duet interpreter by all errors. Can be changed to a positive number, signifying:

5.2.3 Procedure 'init_duetmaskine';

exit to the program point; or changed to a negative number, signifying: continue after the error.
(cf. 'select', section 3.2.14).

max_instruction := 8 388 607;

Is the upper limit for the number of Duet instructions which can be executed within one activation of the Duet interpreter: If the activation of Duet names exceeds this number the Duet system presupposes an indefinite loop in the Duet program. This control, however, will only have any influence if the control program redefines the max_instruktion' to a lower number.

std_assign := 0

when std_assign = 0 no standard value is assigned when reading the standard mark. It can be changed to 1 by the control program or by the Duet operation 'select', if a standard value assign is requested.
(cf. 'read', section 3.2.11).

5.2.3 Procedure 'init_duetmaskine';

```
instruktions_tal := 0;
```

the instruction counter is reset to zero. After exit from the Duet interpreter it can be read, from this variable, how many Duet instructions have been executed.

5.2.4 Procedure 'close_duet';

The 'close_duet' procedure must be called by the control program before the termination of the run, i.e. after the last exit from the Duet interpreter.

It closes all files in the database, and the necessary information about bs files is preserved in the cataloque tail.

Furthermore, the writing of each of the 'prinz' zones in terminated in this way:

- the line buffer is transferred to the zone (thus printing an extra - possibly empty - line),
- end medium is written on the area and
- the zone is closed.

The zone - 'fejlud' must, however, be closed by the control program, as access to that zone might be requested after 'close_duet'.

If the control program wants to provide for the termination of the 'prinz' zones the procedure of
close_soda the Soda system 'close_soda' can be called instead of 'close_duet'.

5.3 Error Procedures

In all error situations the Duet interpreter activates one out of three error procedures, namely 'duet_data_fejl', 'duet_program_fejl' or 'duet_system_fejl' depending on whether the situation is caused by an error in the input, in the Duet program or in the surrounding system.

The error procedure prints a message on the selected error message zone, after which it either returns or exits from the Duet interpreter or exits to a program point in the Duet program. (see section 3.2.14, 'select' and section 5.2.3, 'init_duetmaskine').

5.3.1 Duet Data Error

The procedure 'duet_data_fejl' is called in case of an error in input. It calls the print out procedure 'print_data_error'. This procedure is declared in the outmost block, but it can be redefined by the control program on another block level (cf. section 5.1), if the error messages are required in another language, e.g. Danish.

The standard format of data error messages is:

```
D<NR>. <error text>

***** <error text> { <error parameter> } 2
                                        1
```

In the error messages originating from the 'read' operation, the first error parameter is always the name of the reading variable, which is stored in 'læsvar_navn'.

Below, the error messages are described in alphabetical order:

No decimals in <var_name>

D3: DECIMALS MISSING IN <var_name>

A decimal point has been read after a numerical field, but no decimals are read.

D<NO> DATA ERROR, NOT IMPLEMENTED

Print_data_error kaldt med "fejlnr" > 14

5.3.1 Duet Data Error

Line too short - last field is <var name>

D7 ILLEGAL DELIMITER AFTER <var_name>

The last read field has not been terminated with an ordinary delimiter as required in the reading specification.

Illegal termination of <var. name>

D9 ILLEGAL DELIMITER OR LINE TOO LONG AFTER <var_name>

The last read field is neither terminated with an ordinary delimiter nor an end of string delimiter. (line feed)

Syntax error in field after <var. name>

D1 ILLEGAL FIELD AFTER <var_name>

The contenst of the data field after the last correct field does not correspond to the type in the stated read specification(s).

illegal value of <varname>: <value>

D10 +14 ILLEGAL VALUE FOR <var_name> <value>

↑ D10 : heltal (ut decimaler)
D14 : real

The value of the read field lies outside the value spectrum allowed for the reading variable in question. <value> states the value read for a numerical field, or the number of characters read for a text field.

5.3.1 Duet Data Error

line too long - last field should be <var_name>

D8 | LINE TOO LONG AFTER <var_name>

The last read field has not been terminated with an 'end of string' delimiter (line feed) as required in the read specification.

<set name> does not exist

D11 | RECORD WITH THE SPECIFIED KEY DOES NOT EXIST <set_no>

Unknown record attempted read with 'get'

<set name> exists already

D12 | RECORD WITH THE SPECIFIED KEY PREVIOUSLY CREATED <set_no>

The record is rejected by put after 'create'

content missing in <var name>

D2 | STANDARD MARK ILLEGAL FOR <var_name>

A standard mark has been read to a variable, which does not allow standard value.

System error - please avoid this line code

D13 | SYSTEM ERROR: OPERATION P.T. ILLEGAL

This Duet data error is always activated after a Duet program error or a Duet system error.

5.3.1 Duet Data Error

The text in <var name> not terminated

D5 | TEXT TERMINATION MISSING AT <var_name>

A text field has been interrupted by an end of string delimiter (line feed) prior to the terminating text delimiter (apostrophe).

<var_name> too long - make max <max> chars

D6 | TEXT TOO LONG AT <var_name> <max_length>

The read text field is longer than the specified reading variable
<max_length> states the largest text length allowed.

too many decimals in <varname>, make max <max>

D4 | TOO MANY DECIMALS IN <var_name> <max_decim>

The field in the input has more decimals than the reading variable in question allows.
<max_decim> states the largest number of decimals allowed.

<setname> :

```
<*SETNO   SETNAME*>
<* 1*> <:ITEM MASTER:>,
<* 2*>  <:REFERENCE ITEM:>,
<* 3*>  <:ITEM:>,
<* 4*>  <:PART ITEM:>,
<* 5*>  <:PARTS LIST REF:>,
<* 6*>  <:CUSTOMER MASTER:>,
<* 7*>  <:HIGH LEVEL CUSTOMER:>,
<* 8*>  <:CUSTOMER HEAD:>,
<* 9*>  <:CUSTOMER STRUCTURE REF:>,
<* 10*> <:USER INF:>,
<* 11*> <:ADMIN MASTER:>,
<* 12*> <:WHERE USED LIST REF:>,
<* 13*> <:HIGH LEVEL CUST REF:>,
<* 14*> <:TERMINAL ADM:>,
<* 15*> <:ADMIN EXTRA:>,
<* 16*> <:CUSTOMER:>,
<* 17*> <:ORDER MASTER:>,
<* 18*> <:ORDER LINE:>,
<* 19*> <:VOUCHER HEAD:>,
<* 20*> <:PARENT ORDER LINE:>,
<* 21*> <:ORDER HEAD:>,
<* 22*> <:PART ITEM LINE:>,
<* 23*> <:VOUCHER LINE:>,
<* 24*> <:USER ADMIN:>,
<* 25*> <:ODLINE OF ITEM:>,
<* 26*> <:ORDER LINE COPY:>,
<* 27*> <:ORDER LIST ELEMENT:>,
<* 28*> <:CUSTOMER COPY:>,
<* 29*> <:ACCOUNT ENTRY:>,
<* 30*> <:VOUCHER SCAN LINE:>,
<* 31*> <:DISCOUNT LINE:>,
```

5.3.2 Duet Program Error

The procedure 'duet_program_fejl' is called when the Duet Interpreter detexts any illegal constructions in the Duet program.

The standard format for program error messages is

```
***** PROGRAM ERROR eno: CALL FROM: BLOCK  INSTR.  OPT    (ADR  PIL ART)
                         XXXXXX      XXX    XXX    XXXXXX  XX   XX  XX
     { <error text>  <error parameter> } 3
                                       } 0
```

eno — is the current error number

CALL FROM — is either DUET or the name of a db-operation.

BLOCK, INSTR, OPT — indicates the current block no, Duet name and Duet operation at the errorneous location in the Duet program

ADR, PIL, ART — internal values in the Duet interpreter which may assist the maintenance group in case of errors in the Duet system.

Below the Duet program errors are described in error number order.

1. ILLEGAL BLOCK NO OR BLOCK MISSING <block_no>

Attempt to activate a non-existent Duet block (cf. execute, section 3.2.1).

5.3.2 Duet Program Error

2. ILLEGAL BLOCKREF, USER CONFLICT <block_no> <user_no>

Attempt to activate a Duet block belonging to another user.
<user_no> is the user no of the illegal block. (cf. execute, section 3.2.1).

3. ILLEGAL ENTRY POINT <entry_no>

Attempt to activate a Duet block with entry at a non-existing entry-point. (cf. execute, section 3.2.1).

4. Not used

5. ILLEGAL VARIABLE NUMBER <var_no>

By an indirect variable reference a non-existing variable is referred.

6. INDEX <index_value>

An array variable is referred with an illegal subscript value.

5.3.2 Duet Program Error

7. ILLEGAL CHANNEL NUMBER <channel_no>

With select print a non-existing zone in the zone array 'prinz' is chosen i.e. channel_no > max_kanal or channel_no = 0 for result print-outs (cf. select, section 3.2.14).

8. INSTR. COUNT EXCEEDED <inst_count> <max_inst>

The instruction counter exceeds 'max_instruktion', perhaps due to an infinite loop in the Duet program.

9. ILLEGAL SET NUMBER <set_no>

Illegal set no in db-operation.

10. USAGE CONFLICT <set_no>

db-operation in disagreement with the usage-specification of the set declaration in the ld-description.

5.3.2 Duet Program Error

11. RECORD STATE ILLEGAL FOR THIS OPERATION <set_no> <record_state>

illegal db-operation in relation to the record state of the set.

<record state>:

0	after open
1	after get
2	after put (direct access) or after error
3	after delete (direct access)
4	after create
5	after newset
6	after next
7	after put (sequential)
8	after delete (sequential)
9	after end of chain by delete
10	after end of chain by next
11	after end of chain in an empty chain
12	after end-of-set

12. MOVE ERROR <set_no> <text> <addr> <value>

Error by move between variables and fields in connection with a db-operation.
<addr> and <value> is always the address of and value of the current variable.

<text> is a further explanation of the error cause:

5.3.2 Duet Program Error

SPILL DURING TRANSFER FROM VAR

The value of a variable cannot be contained in the current field.

INDEX

The value of a variable used as subscript is outside the limits of an array or exceeds the current number of repetitions for a repeating group.

ILLEGAL NUMBER OF REPET. IN RPG

Number of repetitions by creation of a repeating group is outside the legal interval.

13. SET CLOSED FOR SEQ. ACCESS BY NEWSET ON ANOTHER SET <set_no>

14. DAUGHTER RECORDS ASSOCIATED WITH CURRENT RECORD

Deletion of a masterfile record is not allowed when the record has any daughter records connected to it.

15. Not used

5.3.2 Duet Program Error

16. CURRENT RECORD REMOVED FROM DB <set_no>

A record read with get or next cannot be found in the file at put or delete.

17. MOTHER RECORD MISSING IN SIDE CHAIN <set_no>

By insertion of a new record into a list file the secondary mother record which was entered by create, is missing.

18. MOTHER RECORD REMOVED FROM DB <set_no>

The mother record for a set (set type L) has been removed after newset.

19. ILLEGAL STATE FOR MOTHER SET <set_no> <record_state>

No current record in the mother set exists in the data base at newset (settype L).

20. ILLEGAL CURRENT RECORD TYPE IN MOTHER SET <set_no>

At newset (settype L) the current record in the mother set is a record type which cannot function as a mother to current set.

5.3.2 Duet Program Error

21. RECORD TYPE ILLEGAL IN SET <set_no> <record_type>

The specified record type at create does not belong to the set.

22. PRINTVALUE EXCEEDS FIELD RANGE <channel> <value>

The number of digit positions is too small for the value to be printed. In the result area the value is printed as '????'.

<channel> indicates the current result zone
<value> is the too big value

23. ILLEGAL PRINTPOSITION <start_pos>

The start position for print is specified outside the line buffer.

24. PRINT FIELD EXCEEDS LINE BUFFER <start_pos> <number>

The calculated last position of a field is outside the line buffer.

5.3.2 Duet Program Error

25. ILLEGAL BS. OPERATION, POSITION AFTER EOF <set_no>

At get or newset (settype B) the specified position is after end-of-file.

26. ILLEGAL DELETE POSITION <set_no>

Current Record in the set (settype B) cannot be deleted because another set in the same file has a current record after this position.

27. SET NOT OPEN FOR SEQUENTIAL ACCESS <set_no>

Newset has not been called or end-of-set is reached.

28. ILLEGAL SPECIAL ACTION <algol_no>

An undefined Algol special-action is activated from the Duet program.

29. ILLEGAL NO OF PARAMETERS <algol_no> <no>

Illegal number of parameters specified for the activated Algol special-action.

5.3.2 Duet Program Error

30. ILLEGAL TYPE OF PARAMETER <algol_no> <param_no>

Illegal parameter type specified for an algol special action.

Add the following new duet program error:

31: illegal zero division

can be caused by modify (integer division) and compute (ordinary division).

5.3.3 Duet System Errors

The procedure 'duet_system_fejl' is called, if the Duet Interpreter detects any inconsistencies in the Duet program, the ld-description, or the database. In case of system errors, the maintenance group should normally be notified.

The standard format for system error messages is:

```
***** SYSTEM ERROR eno:  CALL FROM:  BLOCK INSTR  OPT    (ADR   PIL  ART)
                         xxxxxx       xx    xxx  xxxxxxx  xx     xx   xx
      <errortext> { <errorparameter> } 3
                                       0
```

The information in the system error line is the same as the information in a program error line (see section 5.3.2). Below the system errors are described in error number order.

The information in <> states the names of the variables, which are being printed, as a further indication of the system error.

```
1.  DUETREL <duetrel> <duetstop>
```

There is an inconsistency in the binary Duet program in connection with the activation of a Duet instruction.

5.3.3 Duet System Errors

2. BIT23 = 0

There is an inconsistency in the binary Duet program in connection with the execution of a Duet instruction.

3. 'DUETNAME'

There is an inconsistency in the binary Duet program in connection with the execution of a Duet instruction.

4. VALUE OF 'UDTRYK' <val> <duetstop>

There is an inconsistency in the binary Duet program or an error in the Duet Interpreter's procedure 'udtryk'.

5. BIT23 = 1 <duetstop>

There is an inconsistency in the binary Duet program or an error in the Duet Interpreter's procedure 'udtryk'.

5.3.3 Duet System Errors

6. LD-TABLES <setno>

There is an inconsistency in the binary ld-description.

7. CHECKSUM <setno>

Checksum error in a database record; there is an inconsistency in the database.

8. RECORDLENGTH <setno>

There is an inconsistency in the database: The length of a record which is read with 'get', 'next', or 'lookup' does not correspond to the db-description.

9. CONNECT MISSING IN SIDE CHAIN <setno>

There is an inconsistency in the database: the list file record in question has not been connected to all the secondary mothers.

5.3.3 Duet System Errors

10. FILE EXPANSION IMPOSSIBLE <resultcf>

By put, a file expansion with 'extendcf' has failed.

<resultcf>	states the cause:
2	The file has been extended, but the zone buffer is too small (the variable 'sd_extend_buf' must be initialised with a larger value, if many records are to be inserted in one run).
>10000	Lack of resources.

11. RECORD CREATION TOO EXPENSIVE <resultcf>

Insertion of a new record (put after create) has failed.

<resultcf> states the cause (see ref. 4).

12. DELETION OF LAST RECORD IN FILE

There must always be at least one record in a cf-masterfile.

13. SKIPWORD

There is an inconsistency in the binary Duet program in connection with an exit to a program point.

5.3.3 Duet System Errors

14. OVER/UNDERFLOW <staktrin>

Error in the Duet interpreter by stacking or unstacking. The cause might be too many levels of execute-operations.

15. ERROR IN BLOCK TRANSFER <blockno> <errortext>

Error discovered when reading a Duet block. <errortext> states additional cause:

BLOCKLENGTH ERROR
BLOCKNO ERROR
CHECKSUM ERROR
RECORDNO ERROR

They all state an inconsistency in the binary Duet program

BLOCK/LD VERSION INCONSISTENCY

The Duet block has been compiled with another version of the ld-description than the one the running system operates with.

5.3.3 Duet System Errors

16. MOVE ERROR <text>

An inconsistency arisen during transfer between variables and fields in connection with a db-operation.
<text> is a further explanation of what caused the error:

ILLEGAL ENTRYADDR IN 'FLYTTETAB'

ILLEGAL MOVEINSTRUCTION: <type> <addr>

they both state an inconsistency in the binary ld-description.

17. ILLEGAL TYPE OF VALUE ELEMENT <type>

There is an inconsistency in the binary Duet program or an error in the Duet interpreter's procedure 'take value'.

5.4 Algol special actions

The algol special actions in the Duet program are performed by letting the Duet Interpreter call the procedure 'duet_algol(no)' with the Algol action number as parameter.

duet_algol

This procedure must be declared by the control program at the block level, which 'duet_text2' is copied into.

The procedure may have the appearance shown in fig. 5.3.

```
procedure   duet_algol(no);
value no;   integer    no;
case no of
begin
  begin <*action1*>
    .
    .
    .
  end;
    .
    .
    .
  begin <*actionx*>
    .
    .
    .
  end
end  duet_algol;
```

Figure 5.3: procedure duet_algol

5.4 Algol special actions

Each single Algol action must interpret its parameters itself by repeatedly calling the Duet Interpreter's integer procedure 'takevalue'.

duetparam

The variable 'duetparam' indicates the number of parameters stated, and thus how many times 'takevalue' must be called.

takevalue

The return value of 'takevalue' states how the parameter has been read and the name of the program variables in which the value is stored, as shown in the table fig. 5.4.

takevalue	value type	value stored as an integer	value stored as a floating-point number
0	real	~	reg
1	2	værdi	reg
2	word	d.w, værdi	reg
3	long	d.l, værdi	reg
4	text	d.t	~
5	adr(	procedure 'varadr' has been called	

Figure 5.4 the result of 'takevalue'

varadr

The procedure 'varadr' is called from 'takevalue', when the parameter has been specified as 'adr(...)'. In this case 'type' states where the resulting variable is to be found, as shown in fig. 5.5. Furthermore 'lgd' states, how many words the variable occupies.

5.4 Algol special actions

'type'	variable type	variable is found in	'lgd'
0	text	d.t.	text length in words
1	~	~	~
2	word	d.w	1
3	long	d.l	2
4	real	d.r.	2
5	recno	d.w	1
6	date	d.w	1
7	bits	d.t	aggregate length in words

Figure 5.5. The result of 'varadr'

NB!: If the variable is an array variable it is always the address of the 1. element which is being computed.

goto

Normally an Algol special action must not contain a 'goto' which leads out of the procedure. If the processing of the Duet program is to be continued, the 'duet_algol' must necessarily return, to where it was called from. It is allowed, however, to go to a label outside the Duet interpreter and thereby interrupt the Duet program.

error

The Duet program errors no.s 28, 29, and 30 can be called from the 'duet_algol' procedure, if the latter detects any errors in the call of the special action.

duet_programfejl(28);
28: <:illegal actions nr:>
29: <:illegal antal parametre:> <duetparam>
30: <:illegal parameter type:> <paramno>

5.5 Reserved Algol names

Below is an index of all Algol names, which are declared globally by the Duet text.
Furthermore, all names declared by the Soda_dbms are recerved. (cf. ref. 2).

The index is split up into three lists compressing Algol names, which

1) must/may be used by the control program
2) can be used as working variables
3) must not be touched

The variables in the first list are arranged according to their use, while the two other lists are arranged alphabetically.

5.5.1 Application - known Algol names

Names concerning Algol-special actions
(cf. section 5.4 and 3.2.16)

takevalue — procedure takevalue;
is called from Algol-special actions to fetch the parameters of the action.

```
l       long field          l;
r       real field          r;
w       integer field       w;
t       integer array field t;
reg     real                reg;
værdi   long                værdi;
lgd     integer             lgd;
type    integer             type;
```

contain return values from call of 'takevalue', as described in section 5.4. Apart from that they can be used as working variables without any restrictions.

d — integer array d(-3:maxc);
is the basic array, in which all variables, declared in Soda-ld, and all constants are placed. These can be referenced from Algol-special actions by fielding, either with a field variable generated by the Soda-ld compiler, or by one of the standard field variables w, l, r, or t assigned by the procedure 'takevalue'.

duetalgol — procedure duetalgol(no);
Must be declared by the control program for executing Algol-special actions.

duetparam — integer duetparam;
States the number of parameters for an Algol-special action.

5.5.1 Application - known Algol names

Names concerning reading

(cf. section 3.2.10 and 3.2.11)

read_general

```
external procedure read_general (readz, linie, felter,
                                 felttype, linieindex);
```

reads an input line, to be used by the operation read, as described in the Duet operation 'getline'. Usually the reading is coded in the Duet program, but with some type of program it is more convenient to let the control program itself read in the data lines. In these cases the control program must call 'read_general' with the above - mentioned parameters, and the Duet program must not use the operation 'getline'. This concerns e.g. both 'teleop' and 'telescop'.

readz

```
zone 'readz' (128, 1, stderror);
```

must be opened by the control program before calling 'init_duet2' (cf. section 5.2.2). Is used as a parameter in a possible call of 'read_general'.

linie	real array	linie (1:25);
felter	long array	felter (1:150);
felt_type	integer array	felt_type (1:150);
line_index	integer	linie_index;

are used as parameters for 'read_general'. In the array 'linie' the input line is delivered as a text.

slut på readz? Hurdan signaleras "em - mitat?"

5.5.1 Application - known Algol names

alfa

integer array alfa (0:127);
is initialised by 'init_duet2' with the standard character class table for reading (cf. section 3.2.11, specially fig. 3.35). The control program can change the contents of this table and must then call the procedure 'intable (alfa)'.
Each element in alfa must have the format,
character_class shift 12 + character_value

std_assign

integer std_assign;
see section 5.2.3.

Names concerning print ('primula' printing)
(cf. section 3.2.12 and 5.2.1-4).

prinz

zone array prinz (max_kanal, 128+38,1, prinzproc);
Each zone in 'prinz' must be opened to an output area before calling 'init_duet2' (cf. section 5.2.2).

max_kanal

integer max_kanal;
defines the number of zones in 'prinz'. Must be initialised by the control program before calling 'init_duet1'.

max_printpos

integer max_printpos;
defines the length of the 'primula'-system's line buffer in each of the prinz-zones. Is initialised by 'init_duet1' to 132, but can be redefined in the outer block of the control program.

5.5.1 Application - known Algol names

set_primula

```
procedure set_primula (zone_no);
integer (zone_no;
```

utility procedure, which makes use of 'write' possible on the 'prinz'-zones. Before calling 'write' the control program must call

```
end_print (prinz (zone_no));
```

and after 'write' terminate with

```
set_primula (zone_no);
```

kanal

```
integer kanal;
```

defines which 'prinz'-zone is to be used by print. Is set to 1 by the 'init_duetmaskine'. Can be altered by the control program or by the operation 'select', but must not be reset to zero (cf. section 3.2.14 and 5.2.3).

Names concerning the execution of the Duet program

duetareal

```
integer duetareal;
```

Must be initialised by the control program before calling 'init_duet1'. Defines the size (in words) of the Duet array which the Duet blocks are read to. (cf. section 3.1.2 and 5.2.1).

max_ref

```
integer max_ref;
```

defines 'the half life' of the array 'block_use', in which the Duet system notes how often each single Duet block is referenced. Using this registration it is possible to have the most frequently used blocks remain in the core store (in 'duet_array') (cf. section 3.1.2).

5.5.1 Application - known Algol names

To avoid having a block, which, over a period, has been used frequently, but then is not referenced for some time, remain in the core store, the counters in 'block_use' are halved each time 'max_ref' block references have been executed.
'max_ref' is by 'init_duet2' initialised to 20.

instruktionstal
max_instruktion

```
integer instruktionstal;
integer max_instruktion;
```

the instruction counter and its upper limit (cf. section 5.2.3).

Names concerning initialising
(cf. section 5.2)

duetreg_navn
duet_version

```
real array duetreg_navn (1:2);
integer duet_version;
```

must be initialised by the control program before calling 'init_duet1' with area name and version no. of the compiled Duet program.

ldreg_navn
ld_afsnit_num
ld_version

```
real array  ldreg_navn (1:2);
integer     ld_afsnit_nummer;
integer     ld_version
```

must be initialised by the control program before calling 'init_duet1' with area name, section number, and version no of the compiled ld-description.

fejlud

```
zone fejlud (128, 1, stderror)
```

must be opened and closed by the control program to be used for writing the Duetlog and possible error messages, if these are not redirected to 'out'/'prinz'.

5.5.1 Application - known Algol names

init_duet1

integer procedure init_duet1 (z);
zone z;
The initialising procedure in 'duettext1'. It must be called by the control program with 'out' or 'fejlud' as parameter; the 'prinz'-zones must not be used. See section 5.2.1.

init_duet2
init_duetmaskine

integer procedure 'init_duet2';
procedure 'init_duetmaskine';
Initialising procedures in 'duettext2'. See section 5.2.2 and 5.2.3.

close_duet

procedure close_duet;
Termination procedure for closing zones.
See section 5.2.4.

Notice: Variables which must be initialised are also found under print and execution of Duet program.

Names concerning error messages
(cf. section 5.3)

datafejl_akt
datafejl_kanal
programfejl_akt
programfejl_kanal
systefejl_akt
systemfejl_kanal

```
integer array datafejl_akt (1:max_datafejl);
integer       datafejl_kanal;
integer array programfejl_akt (1:max_programfejl
integer       programfejl_kanal;
integer array systemfejl_akt (1:max_systemfejl);
integer       systemfejl_kanal;
```

are initialised by the 'init_duetmaskine'. They can be altered with the 'select' operation or by the control program. See section 5.2.3.

5.5.1 Application - known Algol names

max_datafejl integer max_datafejl;
max_programfejl integer max_programfejl;
max_systemfejl integer max_systemfejl;
The upper limits for the 'fejlakt'-arrays. They must not be assigned by the control program.

print_dataerror procedure print_datafejl (z, error no);
zone z; integer error no;

læsvarnavn long array læsvarnavn(1:4);
fejlværdi long fejlværdi;
contains, when 'print_datafejl' is called, the name of the Soda variable, which was last used for reading, plus the illegal value.

Names concerning tests

testa integer testa;
testb integer testb;
testc integer testc;
testd integer testd;
teste integer teste;
testf integer testf;
testg integer testg;
testh integer testh;
testvariables, which might be assigned, when needed, by the control program and/or by the Duet operation 'select' (cf. section 3.2.14).

testkanal integer testkanal;
defines where the test output is printed (cf. section 5.2.2).

5.5.1 Application - known Algol names

Other names

duetdato
duettid

```
integer     duetdato;
integer     duetid;
```

are assigned with ISO-date and hour for start of run (hour-minute-seconds) by 'init_duet1'.
They are printed on the Duet log.

duetsystem_vers
duetsystem_dato
duetsystem_init

```
integer     duetsystem_version;
integer     duetsystem_dato;
real array  duetsystem_init (1:2);
```

contain version no and date of publishing plus the initials of the person responsible for the current edition of the Duet system texts. Is printed on the Duet log by 'init_duet1'.

systemtest

```
long array  systemtext (1:4);
```

contains the name of the Duet program (from the Duet head), which is printed on the Duet log.

nl
sp

```
boolean     nl;
boolean     sp;
```

are assigned by 'init_duet1' with respectively 'false add 10' and 'false add 32'.

tipotens

```
integer array tipotens (0:6);
```

is assigned by 'init_duet2' with the tenth powers 1, 10, 100, ..., 1000000.

trykbits

```
procedure trykbits (z, number, word);
zone z; integer number, word;
```

prints the contents of the parameter 'word' as a bit pattern. The parameter 'number' states how many bits of 'word', counted from the right that is to be printed.

---------- Fold here ----------

==========

Affix
postage
here

A/S REGNECENTRALEN
Marketing Department
Falkoner Allé 1
2000 Copenhagen F
Denmark

READER's COMMENTS

DUET

RCSL No: 21-V032

A/S Regnecentralen maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback - your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

_____

_____

_____

Do you find errors in this manual ? If so, specify by page.

_____

_____

How can this manual be improved ?

_____

_____

Other comments ?

_____

_____

_____

_____

==========

Please state your position: _____

Name: _____ Organization: _____

Address: _____ Department: _____

_____ _____

_____ Date: _____

Thank you !

RETURN LETTER - CONTENTS AND LAYOUT

Index (continued)

zero_repr A-192
zero representation 3-73
zero_value
zero value statement 3-74

Index (continued)

value A-189
value check 3-62, 3-67
value element 3-31, 3-40, 3-76
value_spec A-193
value specification 3-71, 3-75
value spectrum 3-67
var 3-44, 3-46
vardecl 5-135
variable 2-10, 3-26,3-81, A-197
variable name 3-26, 3-44, 3-75, 3-91
variable number 3-26
variable reference 3-26
variable type 3-27
varref A-197
var_spec A-188
vartext A-193
version 4-101
vertical_spec A-191
vertical specification 3-71, 3-72
vertical tabulation 3-72
værdi 5-171

w 5-171
while 3-59
while_opt A-190
word var 3-28
working register 3-50
write (between print) 5-174

Index (continued)

test_line	A-195
test output (from the Duet interpreter)	3-85, 3-86, 5-177
test output (from the compiler)	4-107
test_value	A-189, A-190, A-195
test value (assign/action)	3-51, 3-57, 3-75
testvar	A-195
test variable (assign/action)	3-51, 3-55, 3-56
test variable (test outputs)	3-88, 5-177
textchar	A-197
textconst	A-197
text constant	3-30, 3-76, 3-91
text delimiter	3-63, 3-65
text field (in input)	3-63
text file (program text)	4-94, 4-97, 4-101
textlength	A-192
text printing	3-72
textvar	3-28, A-197
text variable (textvar)	3-28, 3-65, 3-75, 3-91
tflytproc	5-136
t-layout	3-72
tno	A-187
to_value	A-190
translate	4-105, 4-106
type	5-168, 5-169, 5-171
user	3-19, 4-101
user adaption	2-10
user number	3-19, 3-21, 4-101

Index (continued)

soda_dbms	1-6, 2-9, 3-80, 5-170
soda-ld	1-6, 2-9, 5-134
sodatext1,-2,-3	5-136
source text	4-100, 4-105
spill	3-29
standard layout	3-79
standard mark	3-68, 3-85, 3-88
standard value	3-62, 3-68, 3-81, 3-88,
std_assign	3-69, 5-144, 5-173
std_assign_line	A-195
std_layout	A-193
std_layoutype	A-193
std_line	A-193
suppression specification	3-37, 4-102
syntactical description	A-185
Sysdok lineno.	4-94
Sysdok file	4-94, 4-97, 4-100, 4-103,
system_fejl_akt	5-143, 5-176
system_fejl_kanal	5-143, 5-176
t	5-171
take value	5-168, 5-171
teleop	1-7, 5-133
telescop	1-7, 5-133
term_code	A-191
terminator code	3-64, 3-65
testassign_symbol	A-195
testbit	3-89, A-195
test channel	3-86
test_kanal	5-141
testkanal	5-177

Index (continued)

result area	3-70
result_channel	A-194
result output	3-86
result variable	3-66, 3-69, 3-84
return_line	A-195
right side	A-188
sd_extend_buf	5-139
section number	4-100
select	3-37, 3-69, 3-70, 3-85 (cont.), 5-177
select exit	3-87
select_line	A-194
select_opt	A-194
select print	3-86
select return	3-87
select stdassign	3-88
select test	3-89
selective assign	3-51
selective branching	3-56
set	3-80 (cont.)
set_no	A-194
set_primula	5-174
set_spec	A-194
set type	3-82, 3-83
shortcar	A-197
simple_numvar	A-197
simple_textvar	A-197
simple_var	A-197
simple variable	3-27
simple_wordvar	A-197
size	4-106
s-layout	3-79

Index (continued)

print_spec A-191
prinz 3-70, 3-86, 5-141, 5-146 5-173
program_fejl_akt 5-143, 5-176
program_fejl_kanal 5-143, 5-176
program point 3-35, 3-87, 5-147
program text 4-94
put 3-81
put_opt A-194

quotation mark 3-63

r 5-171
read 3-60
read_general 3-60, 3-62, 5-136, 5-172
reading variable 3-50
read_mode 3-64, A-191
read_opt A-190
read_spec A-190
read specification 3-64, 3-66, 3-67
readvar A-191
readz 3-60, 5-141, 5-172
real working register 3-50
rec no var 3-28
record field 3-81
record type 3-80
reg 5-171
relation 3-54, 3-59, A-189
relopt A-189
reset to zero of numerical array 3-47
resource demands (for the compiler) 4-109

Index (continued)

numerical printing	3-73
numerical value	3-76
numerical variable	3-28, 3-76, 3-91
num_expr	A-188
num_operand	A-189
num_opt	A-189
numvar	3-28, 3-76, 3-91, A-197
oldduet	4-105
paper	4-103
parameter	A-196
parameter_list	A-196
passive program point	3-36
pno	A-187
pos 1	A-193
pos 2	A-193
position	3-71, 3-72, A-192
prepositioned delimiter (in input)	3-67
prim	5-136
primula	3-70, 5-136
principals	3-74, A-192
print	3-70 (cont.)
print_action	A-191
print channel	3-70, 3-85, 3-86
print_data_error	5-136, 5-148, 5-177
print_error_line	A-194
print_line	A-191
print line	3-71, 4-95
print_opt	A-191
print_result_line	A-194

max_datafejl	5-176, 5-177
max_instruction	5-144, 5-154, 5-175
max_kanal	5-173, 5-139
max_print_pos	5-141, 5-173
max_programfejl	5-176, 5-177
max_ref	5-141, 5-174
max_systemfejl	5-176, 5-177
modulo	3-76
modify	3-39 (cont.), 3-76
modify_line	A-188
modify_opt	A-188
modify_symbol	A-188
mod_spec.	A-188
name	3-44, 3-76, 3-91
name_spec	A-188
next	3-81
newduet	4-105
newset	3-81, 3-83
newset_opt	A-194
nl	A-197
n-layout	3-73
no	A-192
no_of_chars	A-191
no_of_lines	A_191
no_of_rep	A-192
normalization	3-28
np	A-197
num_const	A-197
numerical field (in input)	3-63
numerical constant	3-28, 3-76, 3-91
numerical expression	3-49, 3-76

Index (continued)

jobfile for compilation	4-110, 4-111
l	5-171
layout	A-191
layout_param	A-192
layout parameter	3-71, 3-72 (cont.)
layout specification	3-71 (cont.)
layout type	3-71, 3-72 (cont.) 3-75, A-192
ld_sectionno	4-104
ld_afsnit_nummer	5-139, 5-175
ld-description	2-9, 3-81 (cont.)
ldfields	5-136
ldfile	4-104
ld-file	2-9, 4-97, 4-104, 5-139
ld_ident	A-186
ld_no	A-186
ld_reg_navn	5-139, 5-175
ld_version	5-139, 5-175
leftside	A-188
lgd	5-168, 5-169
line buffer	3-70
line feed (in Duet program)	3-15
lineno.	4-94
list	4-102
listing	4-93, 4-94, 4-98
listout	4-102
log print out	4-112, 5-139, 5-178
lookup	3-80
lookup_opt	A-194
læsvar_navn	5-148, 5-177

Index (continued)

get	3-81
getline	3-60
getline_opt	A-190
get_opt	A-194
Goto	3-35, 5-169
horizontal specification	3-71
if	3-54
if_opt	A-189
implicit standard mark	3-68
include	3-37, 4-102,
index	A-197
informal	A-185
init_duet1	5-135, 5-136, 5-138, 5-139, 5-176
init_duet2	3-62, 3-86, 5-135, 5-138, 5-141, 5-176
init_duetmaskine	3-69, 3-70, 3-72, 3-86, 5-135, 5-138, 5-143, 5-176
initials	4-101
insert	4-105, 4-106,
instruktions_tal	5-145, 5-175
integer division	3-76
integer variable	3-28
internal lineno.	4-94, 4-114
itype_no	A-194
itype_spec	A-194

Index (continued)

error_no	A-195
error_type	A-194
execute_line	A-187
execute_list	2-10, 3-13, 3-15, 3-33 (cont.)
execute_opt	A-187
execution of Duet program	3-15
exit	3-36, 3-58
exit_line	A-195
exit_opt	A-195
expansion percentage	4-106
explicit standard mark	3-68
external lineno.	4-94
fejlud	5-139, 5-146
fejlværdi	5-177
field association	3-81 (cont.)
field transfer	3-81 (cont.)
fixed_pos	A-192
fixed_sign	A-192
fixed sign	3-73
for	3-58
form feed (in print)	3-72
form feed (in the Duet program listing)	4-94
for_opt	A-190
fp-parameter	4-98 (cont.)
fp-parameter key-word	4-99
from_value	A-190

Index (continued)

Duet program name	3-17
Duet program number	3-17
Duet program text	4-94
duet_ref	A-187
Duet reference	3-30, 3-33
duet_reg_navn_	5-139, 5-175
Duet stop	3-14,3-33, 3-39, 3-49, 3-51, 3-55, 3-56, 3-70, 3-85
Duet system	1-6
duet_system_fejl	5-147, 5-161
Duet system error	5-161 (cont.)
Duet texts	5-134
duettext1	5-135, 5-136
duettext2	5-135,5-136, 5-137, 5-167
duettext3	5-135, 5-136, 5-137
duettext_name	4-101
duettid	5-178
duet_version	4-104, 5-139, 5-175
editing (program text)	4-94 (cont.)
else_action	A-190
else_line	3-52, 3-56, A-189
end of string	3-60, 3-63
end of string delimiter	3-65
entry_no	3-19
entrypoint	2-9, 3-18, A-186
entry_spec	A-187
error messages (from the compiler)	4-98, 4-14 (cont.)
error messages (from the Duet interpreter)	3-86

Index (continued)

Duet algol	5-135, 5-136, 5-167, 5-169, 5-171
duetareal	5-139, 5-174
Duetarray	2-9, 5-139, 5-174
duet_block	A-186
Duet blocks	2-9, 3-18 (cont.), 4-93, 4-104, 5-174
Duet compiler	1-6, 2-9, 4-93
duet_data_fejl	5-147, 5-148 (cont.)
duetdato	5-178
Duet file	2-9, 4-93, 4-98, 4-104 (cont.), 5-139
duet_head	A-186
Duet head	3-17, 4-93
duet_instr	A-186
Duet instruction	2-10, 3-13
Duet interpreter	1-6, 2-9, 4-93, 5-136
Duet log	4-112,5-139, 5-178
Duet name	3-13, 3-24, 3-30, 3-71
duet_no	A-186
Duet operand	3-14, 3-26 (cont.)
Duet operation	3-10, 3-24, A-187
Duet operator	3-14
duetparam	5-168
duetprg_ident	A-186
Duet program	2-9, 4-97
duetprogram	A-186
Duet program error	5-152 (cont.)
duet_program_fejl	5-147

Index (continued)

constant	3-28
control program	1-6, 1-7, 2-9, 3-22, 3-60, 3-62, 3-69, 3-70, 3-89, 5-133 (cont.), 5-167
create	3-81
create_opt	A-194
current record	3-81 (cont.)
comment delimiter	3-63
d-array	2-10, 3-20, 5-171
database	3-80
data error (data_fejl)	3-66, 3-68, 5-148 (cont.)
data_fejl_akt	5-143, 5-176
data_fejl_kanal	5-143, 5-176
db-description	2-9
db-operation	3-80 (cont.)
db-opt	A-193
decimals	3-73, A-192
declaration file	5-134
delete (db-operation)	3-83
delete (duet block	4-105, 4-106
delete_opt	5-194
delimiter	3-64, 3-65
descripfile	4-104
digit (group)	3-42, 3-76
digitno	A-188
div_spec	A-188
d_name	A-197
Duetabler	1-6, 2-9, 4-93, 5-133
Duet action	3-71

Index (continued)

block_no	A-186
block_number	3-18, 4-105 (cont.)
block_ref	A-187
block reference	3-34
block_spec	A-187
Boss lineno.	4-94
bs-file	3-82, 5-146
case	3-55
case_line	A-190
case_opt	A-190
cf-list file	3-82
cf-master file	3-81, 3-82
change	4-105, 4-106
channel	3-86, 5-174, A-194
char_const	A-197
character constant	3-30, 3-76, 3-91
character field (in input)	3-63, 3-65
char_print	3-73
character printing	3-73
character set table	3-62, 3-63, 5-141
character class	3-63, 5-173
chars	3-65
c-layout	3-73
close_duet	5-135, 5-138, 5-146, 5-176
comment field (in input)	3-63
comment (in Duet program)	3-15
compilation job	4-110, 4-111
compute	3-49 (cont.), 3-76
compute_line	A-188
compute_opt	A-188
conditional compilation	3-37

Index

action	3-56
action_line	A-109
action list	3-56
action_opt	A-189
active program point	3-36, 3-90
adaption point	2-10
adr	3-91
a-layout	3-79
alfa	5-141, 5-173
algol	3-91 (cont.)
algol_opt	A-196
algol special action	3-91, 5-167 (cont). 5-171
alternative action	3-53, 3-56
alternative value	3-53
ap	A-197
apostrophes	3-30, 3-63, 3-65
array_var	3-26, A-197
assign_line	A-189
Assign list	3-52
assign_opt	A-189
assign operator	3-40, 3-41, 3-44, 3-47, 3-49, 3-89
assign_value	A-189
assign variable	3-51
automatic normalization	3-28, 3-43, 3-76
basispos_spec	A-191
basic position	3-71
bit var(iable)	3-28
block_end	3-19, A-186
Block_head	3-19, A-186

Appendix A

Basic concepts:

```
nl              .= <10>  |  ';'  textchar(*) <10>)(+);
np              .= ('-+-'  |  <12>)  nl(*);

d_name          .= 'd'  NUMBER(1 TO 999);

varref          .= simple_var  |
                   array_var  '('  index  ')';

simple_var      .= variable (: no arrayspec);

array_var       .= variable (: arrayspec);

variable        .= 'v'  NUMBER(1 TO max_var)  |
                   var_ident;

index           .= numvar  |
                   NUMBER(1 TO max_index);

numvar          .= varref (: vartype = word, long, real, date);
textvar         .= varref (: vartype = text);

simple_numvar   .= simple_var (: vartype = word, long, real, date);
simple_textvar  .= simple_var (: vartype = text);
simple_wordvar  .= simple_var (: vartype = word);
num_const       .= NUMBER  ('.'  NUMBER)(.));
text_const      .= ap  textchar(*)  ap;
ap              .= <39>;                          <* apostrof:  '  *>
char_const      .= '.'  shortchar  shortchar(.)  shortchar(.)  '.';

shortchar       .= <letter, digit, ',-+:;!?/_=<>*()%&'>
textchar        .= <shortchar,  '.'>
```

Appendix A

```
algol_opt           .= 'algol'  parameter_list(.);

parameter_list      .= '('  parameter (','  parameter)(*)  ')';

parameter           .= num_const   |
                       text_const  |
                       char_const  |
                       varref      |
                       name_spec   |
                       adr_spec;

name_spec              see modify_opt.

adr_spec            .= 'adr'  '('  (variable  |  var_spec)  ')';

var_spec               see modify_opt.
```

Appendix A

```
print_error_line    .= 'print'  (error_type | 'test')  'on'  channel;

  error_type        .= 'data'  'error'     |
                       'program'  'error'  |
                       'system'  'error';

  channel           .= NUMBER(0 TO 9);

exit_line           .= 'exit'  pno  'on'  error_type
                       (error_no  (','  error_no)(*))  (.);

  pno                  see execute_opt

  error_no          .= NUMBER(1 TO max_error);

return_line         .= 'return'  'on'  error_type;

test_line           .= 'test'  testvar  testassign_symbol  test_value;

  testvar           .= 'a'  |  'b'  |  'c'  |  'd'  |
                       'e'  |  'f'  |  'g'  |  'h';

  testassign_symbol .= ':='  |  ':+'  |  ':-';

  test_value        .= numvar  |  'on'  |  'of'  |
                       testbit  (','  testbit)(*);

  testbit           .= NUMBER(0 TO 23);

std_assign_line     .= 'no'(.)  'stdassign'  'on'  'read';


exit_opt            .= 'exit'  pno;

pno                     see execute_opt
```

Appendix A

```
create_opt          .= 'create'  set_spec  itype_spec;

lookup_opt          .= 'lookup'  set_spec;

get_opt             .= 'get'  set_spec;

put_opt             .= 'put'  set_spec;

delete_opt          .= 'delete'  set_spec;

newset_opt          .= 'newset'  set_spec;

set_spec            .= 's'  set_no  |
                       's('  simpel_wordvar  ')';

set_no              .= NUMBER(1 TO max_set);

itype_spec          .= 'i'  itype_no  |                    <*record type*>
                       'i('  simpel_wordvar  ')';

itype_no            .= NUMBER


select_opt          .= 'select'  select_line(*)  's';

select_line         .= nl  |
                       (print_result_line  |
                        print_error_line   |
                        exit_line          |
                        return_line        |
                        test_line          |
                        std_assign_line ) nl;

print_result_line   .= 'print'  'on'  result_channel;

  result_channel    .= NUMBER(1 TO 9);
```

Appendix A

```
value_spec      .= IF layouttype = 't' THEN
                       (text_const !
                        textvar    !
                        name_spec)
                   ELSE
                       (num_const  !
                        char_const !
                        numvar     !
                        num_expr   !
                        mod_spec   !
                        div_spec   !
                        digits_spec);

name_spec
mod_spec            see modify_opt
div_spec
digits_spec

num_expr            see compute_opt

std_line        .= stdlayout vartext simpel_var;

std_layout      .= '<' std_layouttype pos1 ',' pos2 '>';

std_layouttype  .= 's' ! 'a';

pos1            .= fixed_pos;

pos2            .= fixed_pos;

vartext         .= IF std_layouttype = 's' THEN textkonst;

db_opt          .= create_opt !
                   lookup_opt !
                   get_opt    !
                   put_opt    !
                   delete_opt !
                   newset_opt ;
```

Appendix A

```
position        .= fixed_pos | '(' simp_wordvar ')';

fixed_pos       .= NUMBER(1 TO 127);

layouttype      .= 'n' | 't' | 'c';

layoutparam     .= TABLE layouttype = (
                   't' ⊄ textlength,
                   'c' ⊄ no_of_rep,
                   'n' ⊄ fixed_sign(.) zero_repr(.)
                         zero_value(.) principals
                         ('.' decimals)(.)      );

textlength      .= no | '(' simp_wordvar ')';

no_of_rep       .= no | '(' simp_wordvar ')';

no              .= NUMBER(1 TO 127);

fixed_sign      .= '-';

zero_repr       .= '*' | 'z';

zero_value      .= 'b';

principals      .= NUMBER(1 TO 15);

decimals        .= NUMBER(1 TO 7);
```

Appendix A

```
read_mode        .= 'n'  term_code  |
                    't'  term_code  |
                    'c'  term_code    no_of_chars;

term_code        .= NUMBER(0 TO 3);

no_of_chars      .= NUMBER(1 TO 3);

read_var         .= IF  read_mode = 't'  THEN  textvar
                    ELSE  numvar;


print_opt        .= 'print'
                    (print_action  |  print_line  |  std_line)(*)  's';

print_action     .= d_name  nl;

print_line       .= layout  value_spec(.)  nl;

layout           .= basispos_spec  |  print_spec;

basispos_spec    .= '<'  'h'  position  '>';

printspec        .= '<'  verticalspec(.)
                    (position  layouttype  layoutparam)(.)  '>';

  verticalspec   .= 'p'  |
                    'w'  |
                    (no_of_lines  |  simp_wordvar)  'l';

  no_of_lines    .= NUMBER(1 TO 70);
```

Appendix A

```
action_line     .= test_value ':' d_name nl;

test_value      .= num_const | char_const | simp_wordvar;

else_action     .= 'else' d_name;

case_opt        .= 'case' simp_wordvar 'of'
                   case_line(*) (else_action | 's');

case_line       .= (caseno ':')(.) d_name nl;

else_action        see action_opt.

for_opt         .= 'for' simp_wordvar ':=' from_value ',' to_value
                   'do' d_name nl;

from_value      .= num_const | char_const | numvar;

to_value        .= num_const | char_const | numvar;

while_opt       .= 'while' relation 'do' dname;

relation           see if_opt

getline_opt     .= 'getline' simpel _textvar;

read_opt        .= 'read' read_spec (',' read_spec)(*);

read_spec       .= read_mode read_var;
```

Appendix A

```
num_operand     .= num_const   |
                   char_const  |
                   numvar      |
                   '('  numexpr  ')';

num_opt         .= '-'  |  '+'  |  '*'  |  '/';

assign_opt      .= 'assign'  simp_wordvar  ':='  simp_numvar  'of'  nl
                   assign_line(*)  (else_line  |  's');

assign_line     .= test_value  ':'  assign_value  nl;

test_value      .= num_const  |  char_const  |  simp_wordvar;

assign_value    .= num_const  |  char_const  |  simp_wordvar;

else_line       .= 'else'  (assign_value  |  d_name);

if_opt          .= 'if'  relation  'then'  d_name
                   ('else'  d_name)(.);

relation        .= numvar  relopt  value;

relopt          .= '<'  |  '>'  |  '<>'  |  '=' ;

value           .= num_const  |  char_const  |  numvar;

action_opt      .= 'action'  simpel_numvar  'of'  nl
                   action_line(*)  (else_action  |  's');
```

Appendix A

```
modify_opt      .= 'modify'  modify_line  's';

modify_line     .= (leftside  modify_symbol  rightside)(.)  nl;

leftside        .= varref  !  array_var;

modify_symbol   .= :=  !  :+  !  :- ;

rightside       .= IF leftside = array_var THEN
                       (array_var  !  0 )
                   ELSE
                     (varref        !
                      num_const     !
                      text_const    !
                      char_const    !
                      mod_spec      !
                      div_spec      !
                      digits_spec   !
                      name_spec     !
                      var_spec       );

mod_spec        .= numvar  'mod'  (numvar  !  num_const);

div_spec        .= numvar  '//'  (numvar  !  num_const);

digits_spec     .= 'digits'  '('  digitno  ','  digitno  ')'
                   'of'  numvar;

digitno         .= NUMBER(1 TO 15);

name_spec       .= 'name'  '('  (varref  !  var_spec)  ')';

var_spec        .= 'var'  '('  (numvar  !  var_spec)  ')';

compute_opt     .= 'compute' compute_line(*)  's';

compute_line    .= (numvar  ':=')(+)  num_expr;

num_expr        .= '-'(.)  num_operand  (num_opt  num_operand)(*);
```

Appendix A

```
duet_operation      .= (execute_opt   !
                        modify_opt    !
                        compute_opt   !
                        assign_opt    !
                        if_opt        !
                        action_opt    !
                        case_opt      !
                        for_opt       !
                        while_opt     !
                        getline_opt   !
                        read_opt      !
                        print_opt     !
                        db_opt        !
                        select_opt    !
                        exit_opt      !
                        algol_opt      )  nl;

execute_opt         .= 'execute'  execute_line(+)  's';

execute_line        .= (tno(.)  pno(.)  duetref  (',' duetref)(*))(.)  nl;

tno                 .= 't'  NUMBER(1 TO 9);

pno                 .= 'p'  NUMBER(1 TO 9);

duetref             .= d_name  !  block_ref;

block_ref           .= 'b('  block_spec  ','  entry_spec  ')';

block_spec          .= block_no  !  simpel_numvar;

entry_spec          .= entry_no  !  simpel_numvar;
```

Appendix A

```
duetprogram     .= duet_head  duet_block(*);

duet_head       .= duetprg_ident  nl  ld_ident  nl;

duetprg_ident   .= 'duetprogram'  duet_no  '/'  textconst;

duet_no         .= NUMBER;

ld_ident        .= ('localdata'  ld_no)      !
                   ('dataentry'  'ld'  ld_no);

ld_no           .= NUMBER;

duet_block      .= block_head  duet_instr(*)  block_end;

block_head      .= 'begin'  block_no  ':'  nl
                   entrypoint(*);

block_no        .= NUMBER (1 TO 255);

entrypoint      .= '->'  entry_no  '='  d_name  nl;

entry_no        .= NUMBER(1 TO 63);

block_end       .= 'end'  block_no  nl;

duet_instr      .= nl(*) np(.)  d_name  ':'  duet_operation
                   (nl(.)  '>'  duet_operation)(*);
```

Appendix A A Syntactical Description of the Duet Language

This appendix contains a formal syntactical description of the Duet language. The syntax is described by means of the Informal language (ref. 7), but modified so that the (concatenating symbol -* is replaced by space.

Below is an index of the symbols of the Informal language, and their meaning. For a more detailed explanation see ref. 7 and Appendix A of the Soda-manual (ref. 2).

.=	definition symbol for syntactical unit (SU),
;	termination of statement,
!	alternatives,
(.)	SU may be omitted or stated only once.
(*)	SU may be omitted or stated an arbitrary number of times.
(+)	SU must occur at least once
' '	SU is a constant text string
< >	contents of a character set
(:)	restriction-symbol for data quantity

In the following syntactical description all basic concepts are defined at the end while all other syntactical units are defined immediately after the position where they are mentioned the first time.

5.5.3 Inaccessible Algol Names

integer	refantal
integer	reftal
integer field	resultat1
integer field	resultat2
integer field	resultat3
integer field	resultat4
integer field	resultat5
label	skipterm
label	slut_læs
label	slut_short
procedure	stak
integer	stakpunkt
integer	staktrin
integer	stopadr
procedure	takenum
label	test_relation
procedure	tryk_identid
label	tryk_vertikal
integer array	typelængde
label	udfør
procedure	udtryk
procedure	varadr
integer array field	vartextabase
procedure	varværdi
procedure	værdigrænsekontrol

5.5.3 Inaccessible Algol Names

integer	exitnr
long	fejlværdi
integer	fejlindex
procedure	flytblok
label	fortsæt
label	fortsæt_execute
label	fortsæt_layout
label	fortsæt_modify
label	genoptab_instruktion
integer	gl_duetnavn
integer	index
boolean	in_if
integer	instruktionstal
integer array	klumpbasis
integer array	klumpref
integer	lokalbase
integer	læsvar
integer	maxblok
integer	maxd
integer	maxklump
integer	maxstak
integer	maxvar
integer	maxvartext
procedure	nyblok
label	perform
integer	position
integer	primula_state
procedure	print_duet-test
procedure	print_flyttefejl
procedure	print_programfejl
procedure	print_systemfejl
blockprocedure	prinz_proc
integer	punktblok
integer	punktpil
integer array	punktstak

5.5.3 Inaccessible Algol Names

procedure	afstak
label	aktion_fundet
integer array	basispos
label	bestem_element
integer array	blokbasis
integer array	blokbrug
integer	blokbruger
integer	bloklgd
integer array	bloklængde
integer	blokno
integer	bloknr
integer array	blokplac
integer	blokslut
label	checksoda
procedure	checkværdi
integer	dadr
integer	duetadr
integer	duetart
integer	duetbasis
procedure	duetdatafejl
integer	duetnavn
integer	duetoperator
integer	duetord
integer	duetpil
procedure	duetprogramfejl
integer	duetpunkt
integer	duetrel
long array	duetstak
integer	duetstop
procedure	duetsystemfejl
procedure	duettest
zone	duetz
label	element_fundet
integer	entrynr
label	exit

5.5.2 Free working variables

```
integer                 spec
integer                 stop
integer                 sætnr
long                    talværdi
integer                 term
integer                 termkrav
integer array field     text
integer                 til
integer                 vadr
integer                 val
integer                 var
long                    varinf
integer                 varnr
long                    varord
integer                 word
```

5.5.2 Free working variables

```
integer                adr
integer                adrtype
integer                aktion
integer                antal
boolean                b
integer                fejlbits
integer                fejlnr
integer                formatinf
integer                fra
integer                ftype
long array field       haf
integer                i
integer array          ia (1:20)
integer array field    inf
integer                j
integer                k
integer                klasse
integer                kode
long array field       laf
long                   lword
integer                minus
integer                mode
integer                norm
integer                nr
integer                opd
integer                opdtype
integer                operator
integer                opord
integer                opt
integer                pil
real array field       raf
real                   reg1
```

5.5.1 Application - known Algol names

takenameReal procedure takename (ra);
real array ra;
simulates 'increase' in a way which does not require a subscript as a parameter. (Is identical with the procedure 'hentnavn' in 'begin80', but has been given its own name, in order not to bother those users who employ 'begin80' in the outer block of the control program).