


---

Title:

DOMUS, System Programmer's Guide

---

 **REGNECENTRALEN**

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

---

RCSL No: 43-RI0164  
Edition: February 1977  
Author: Philippe Gaugin

---

Keywords:

MUS, Operating System, Loader, Disk

---

Abstract:

This manual describes the interface between assembly program and DOMUS.  
22 pages.

CONTENTS

PAGE

1.	INTRODUCTION	1
2.	CORE STORAGE STRUCTURE	2
2.1	Core Items	2
2.2	Process Hierachy	5
3.	PROGRAM AND PROCESSES	6
3.1	Program Descriptor Word	6
3.2	Program Parameters	6
3.2.1	Separators	7
3.2.2	Item types	8
3.2.3	Scanning the Parameters	10
3.2.4	Parameter Example	11
4.	COMMUNICATION WITH S	12
4.1	Formats	14

APPENDIX A, SYNTAX OF S-COMMANDS

APPENDIX B, SURVEY OF S-COMMANDS

APPENDIX C, SURVEY OF ERROR MESSAGES AND  
EQUIVALENT ERROR NUMBERS

REFERENCES

## 1. INTRODUCTION

The DOMUS-system consists of the following software components:

- Monitor
- Utility Procedures
- Basic i/o
- Character i/o
- Record i/o
- Paging System
- Disc driver
- Teletype driver
- File Management System
- Operating System S

The operating system S takes care of core storage management, program/process load from disc files, program/process removal. It executes commands keyed in from the teletype or send to the process from another process in the system. It introduces a process hierachy among the processes defining a parent/child relation between processes.

## 2. CORE STORAGE STRUCTURE.

### 2.1 Core Items

Programs and processes are organized as described in [1]. After system initialization the available core storage above the basic system is organized as one large item of core storage. When the system works, pieces of that core storage is occupied by additional programs, procedures, processes or data. Core storage is allocated in disjoint pieces, called core items, all chained together in ascending order in the one and same chain, the core item chain, the head of which being addressed by the page zero location: CORE.

The core item is headed by a 7 word descriptor with the following contents

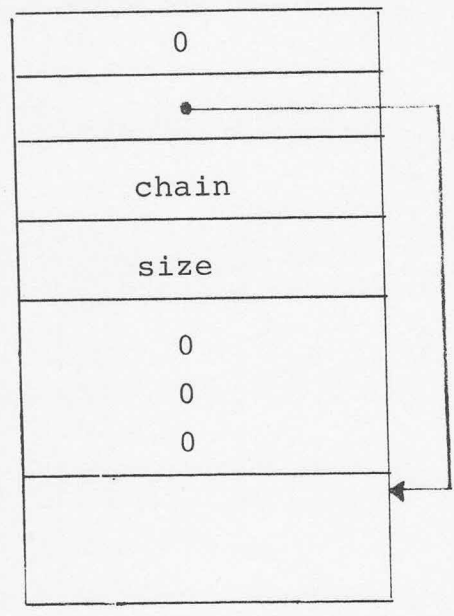
+0	owner process
+1	current load address
+2	chain
+3	size
+4	name of core item
+5	
+6	

The owner process field contains the process description address of the process which allocated the item. The current load address field points to the first address to be used if the core item is boaded with a procedure or a process. The chain field points to the next core item or is set to zero if it is the last item in the chain. The size field contains the size of the core item. The name field contains a possible name of the core item.

Three different kinds of core items exists:

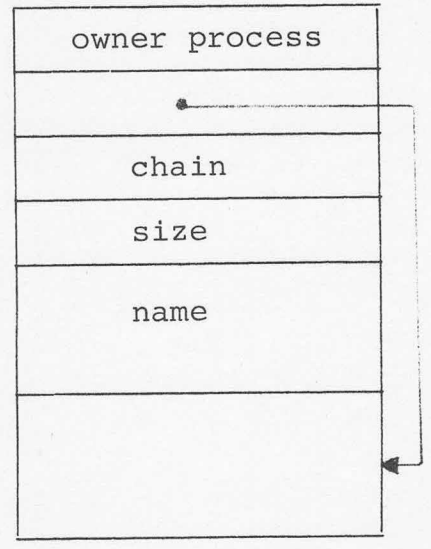
Free core item:

The item is not used by any process.



Used core item:

The item is allocated by a process.



Utility core item

The item is allocated because of a single load

owner process
0
chain
size
name of load file

All processes except for the processes in the basic system are contained in exactly one core item. In a core item may reside several processes. Thus two relations exists between a process  $p$  and a core item  $C$ .

1)  $p$  in  $C$   $\equiv$  }  $p$  lies inside the core item  $C$   
 $C$  contains  $p$ : }

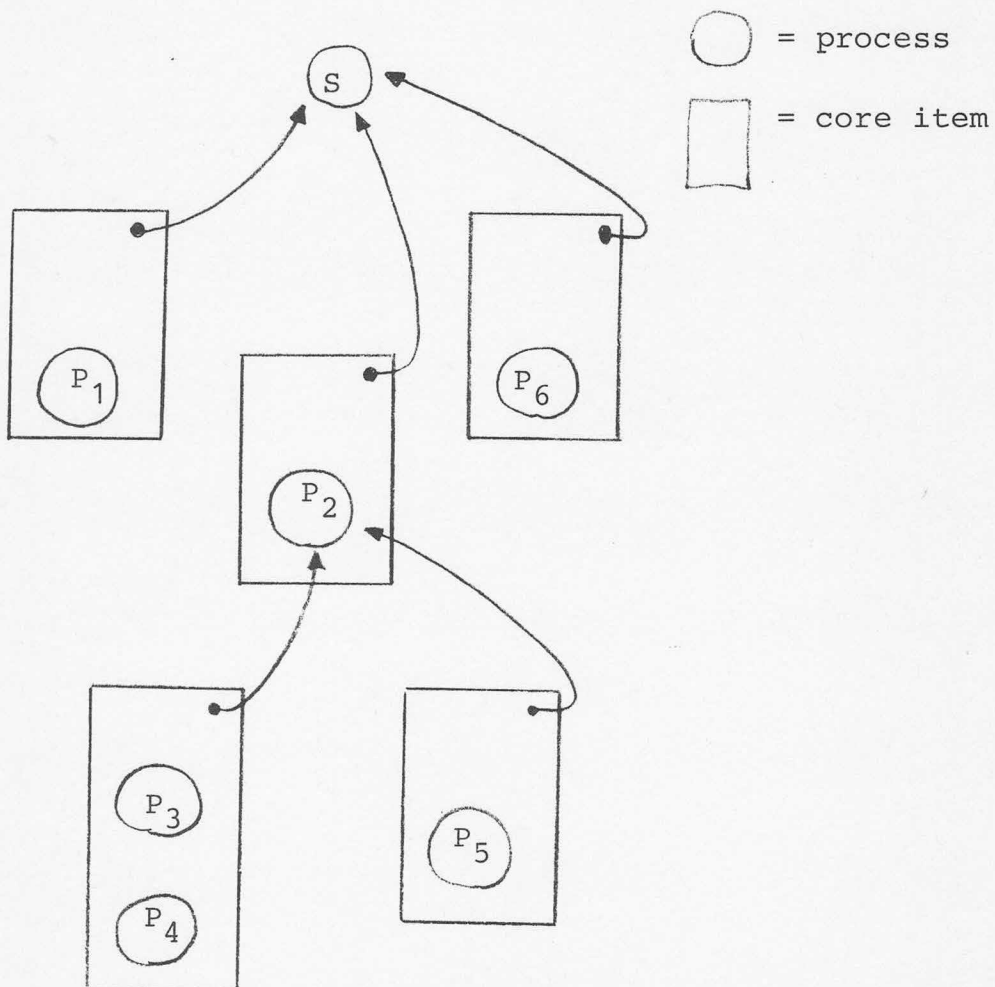
2)  $p$  owns  $C$   $\equiv$  }  $C$ 's owner is  $p$   
 $C$  owned by  $p$ : }

## 2.2 Process Hierachy

The above mentioned two relations between core items and processes introduce a relation between processes:

$$\begin{array}{l}
 p_1 \text{ parent to } p_2 \equiv \\
 p_2 \text{ child of } p_1:
 \end{array}
 \left. \vphantom{\begin{array}{l} p_1 \text{ parent to } p_2 \\ p_2 \text{ child of } p_1 \end{array}} \right\} \begin{array}{l}
 \text{there exists a core item } C \text{ so that} \\
 p_2 \text{ in } C \text{ and } C \text{ owned by } p_1
 \end{array}$$

All processes except for the processes in the basic system are children of other processes and all these processes are organized in a structure with respect to the relation parent to.





### 3. PROGRAMS AND PROCESSES

#### 3.1 Program Descriptor Word

The following bits of pspec are used by the system:

- B5: parameter bit: if this bit is set, parameters are placed immediately after the highest location occupied by the code. If the program contains a process descriptor the address of the parameter are placed in ac 1 when the process is started.
- B6: paged program bit: the program is paged  
see ref [3]
- B7: reservation bit: the process is a driver process. Processes using this program are broken with cause = 8 if a reserver of the process is killed.

#### 3.2 Program Parameters

Program parameters are placed immediately after the loaded relocatable code, and consists of a sequence of items, each of the form

LENGTH	
SEP	TYPE
PARAMETER	

An item always start at an even byte address. The length of an item is the number of bytes in the item, odd or even. The separator is a byte containing the ascii value of the separator preceeding the item or a zero if the item was preceeded by no separator. The type is an integer denoting the type of the item.

### 3.2.1 Separators

The following values may appear in the separator field:

0:	blank, no separator.
44:	,
46:	.
47:	/
58:	:
61:	=

### 3.2.2 Item Types

Type 0, names:

10	
SEP	0
char1	char2
char3	char4
char5	0

In a name item up to five characters of the name are placed, unused positions being zeroised. Only the first five characters of a name are significant.

Type 1, integer:

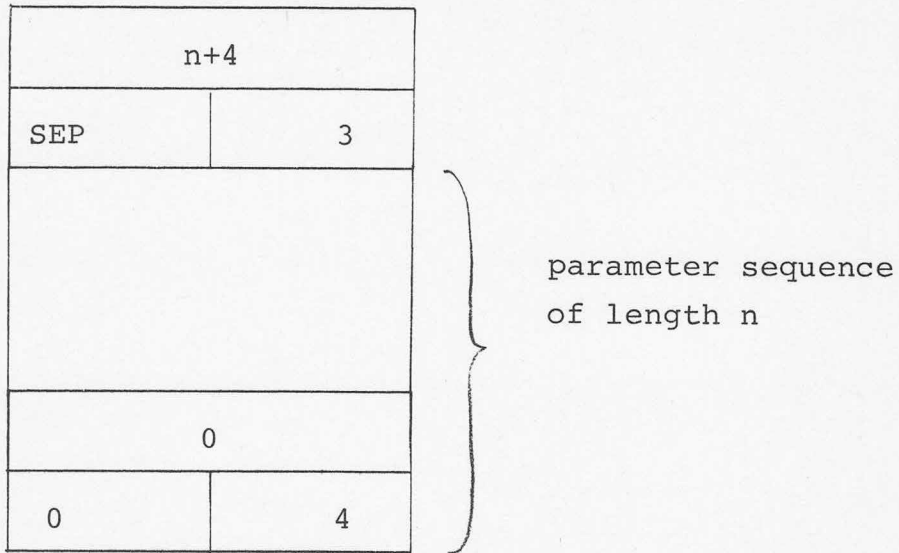
6	
SEP	1
value of integer	

Type 2, texts:

n+4	
SEP	2
char1	char2
...	...
char <sub>n-1</sub>	char <sub>n</sub>

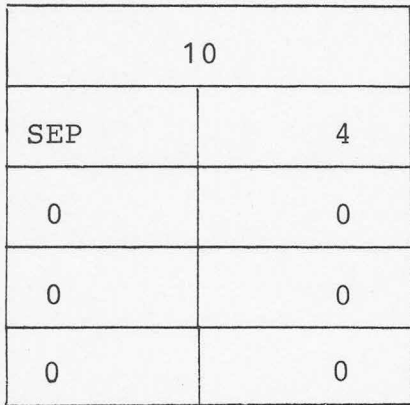
If n odd the slack byte is set to zero

Type 3, composite item:

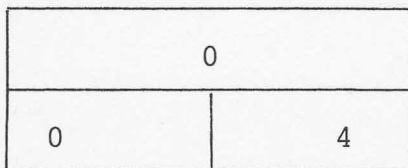


The composite item contains all parameters enclosed in parentheses. The sub sequence starts in  $\text{adr}(\text{item})+2$ .

Type 4 dummy item:



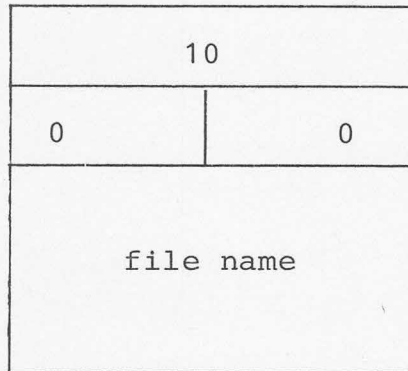
Type 4 end item:



Note that the end of a parameter sequence is signified by a dummy item type and by length = 0.

### 3.2.3 Scanning the Parameters

The first item contains the name of the filename from where the program was loaded:



The next item is found using the following algorithm:

```

; ac2 = adr(item)
FNP:                ; FETCH NEXT PARAMETER:
    LDA    0    0,2 ;    length:=item.length;
    INCZR  0,0    ;    size:=(length+1)/2;
    ADD    0,2    ;    adr:=adr+size;
; ac2 = adr(next item)

```

Note that the end of the list is found when the length equals zero. Also note that this algorithm will never pass beyond the end item, but will continuously give you the end item.

3.2.4 Parameter Example

COMMAND: PIP 987/'AB'

PARAMETERS:

10	
0	0
80	73
80	0
0	0
6	
0	1
987	
6	
47	2
65	66
0	
0	4

#### 4. COMMUNICATION WITH S

After system initialization the process S goes into its idle state where it is waiting ready to execute an S-function. As an event arrives, S classifies it as being one of the following types of events.

##### 1. Console command

An answer arrives from the teletype indicating that a human operator wants S to perform an S-function.

##### 2. Internal command

An output message from a process in the system arrives indicating that a process wants S to perform a number of S-functions.

##### 3. Internal request

An control message from a process in the system arrives indicating that a process wants S to perform a final operation on the process itself.

##### 4. Get message

An input message from a process in the system arrives indicating that a process in the system wants S to deliver a message from the message file.

When executing console commands, the operating process is defined as S itself. When executing internal commands the operating process is the sender of the message. Internal requests are only executed if the sender of the message is a child of S, and the operating process is then defined as S itself.

Generally S accepts to operate only on core items owned by the operating process. So the only processes S would kill is the children of the operating process. Some special functions however violate these rules. In [2] all functions are described. In appendix B a survey of commands are listed.

Errormessages consists of 3 components

1. An error cause
2. Possibly a name
3. Possibly a number

When printing errormessages on the teletype the errormessage would look like:

```
***<text> [<name>] [<number in octal>]
```

where the text explains the error cause.

When returning answers to internal messages the errorcause is represented as a positive number.



4.1 FormatsInternal command

Message:

3
COUNT
ADDRESS
NAMEADDRESS

The count is the number of characters to be interpreted by S  
 The address is the byte address of the first character in the message. The name address is a byte address printing to a 6 byte core storage area or zero if no area present.

If <command string> denotes the contents of the bytes in locations address, address+1, ..., address+count-1, the command is interpreted as if the following commands were keyed in from the teletype:

```
BEGIN <nl><command string><nl>END<nl><end medium>
```

The scan terminates when the first END is met. Note that if the bytecount is accurate you need not put an END as the last command in the command string.

Answer:

RESULT*256
COUNT
NUMBER
NAMEADDRESS

The result is zero if the commands were executed successfully otherwise if contains a positive errorcode, corresponding to an errortext. The count is equal to the count of the message. The number is the number part of the message irrelevant if zero. If the name field exists and contains a non null name, that name is the name part of the errormessage. Be careful not to send a message to S with an undefined value of mess 3.

Internal request

Message:

1B8
COUNT
ADDRESS
NAMEADDRESS

The count and address acts as for internal commands. The command string is interpreted as if the following commands were keyed in from the teletype:

```
BEGIN<nl>KILL<sender><nl>
<command string><nl>END<nl><end medium>
```

Answer:

RESULT*256
COUNT
NUMBER
NAMEADDRESS

The answer is given immediately and the process will be removed as fast as possible.

Get message

Message:

5
COUNT > 32
ADDRESS
MESSAGENUMBER

This message asks for a transfer of the text of the system message with the number given in mess3.

Answer:

RESULT*256
COUNT
NUMBER
0

The count is the number of characters in the text including a terminating zero byte. If count of message is too small, only a part of the text is delivered.

## APPENDIX A, SYNTAX OF S-COMMANDS

The following metalinguistic symbols are used:

Sequences of characters enclosed in < and > represent metalinguistic variables whose values are sequences of symbols. The mark ::= means "may be composed of" and the mark | means "or". The production (rule): <sign> ::= + | - means that any occurrence of the variable <sign> may be replaced by a + or a -. The braces { and } signifies that the contents should be regarded as a single metalinguistic variable. The superscription \* means zero or more occurrences of the preceding variable, whereas the superscription + means one or more occurrences. The brackets [ and ] indicates an optional string.

```
<command>      ::= <nl>* <name> { [<sep>] <item>}* <nl>
<item>         ::= <name> | <number> | <text> |
                  <dummy item> | <composite item>
<composite item> ::= (<item> { [<sep>] <item> }*)
<name>         ::= <letter> { <letter> | <digit>}*
<number>       ::= [<sign>] {<digit>+}* <digit> +
<text>         ::= '<any character except>'
<dummy item>   ::= *
<sep>          ::= . | / | : | = | ,
<sign>         ::= + | -
<letter>       ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N |
                  O | P | Q | R | S | T | U | V | W | X | Y | Z | Æ | Ø | Å |
                  $ |
                  a | b | c | d | e | f | g | h | i | j | k | l | m | n |
                  o | p | q | r | s | t | u | v | w | x | y | z | æ | ø | å
<digit>        ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<nl>           ::= ascii characters LF,VT,FF or CR
```

The ascii characters: SP and HT and the sequence ! <any characters except !>! are blind outside texts, except for being terminators of names and numbers.

## APPENDIX B, SURVEY OF S COMMANDS

BEGIN	Read a sequence of command lines from the teletype. Terminate at an END command.
BOOT <filename>	Bootstrap a stand-alone program.
BREAK <process>	Break the selected process.
CLEAN <process>	Stop and clean the selected process.
CLEAR [ <u>&lt;coreitem&gt;</u> ]	Kill all processes in the selected coreitem or in all utility coreitems.
CREATE <file><type><size>	Create a file on the current drive.
DELETE <file>	Delete the selected file.
DRIVE <driveno>	Select the specified drive as current drive.
END	Terminate a sequence of commands and execute these commands.
FREE <coreitem>	Free the specified coreitem.
GET <coreitem> [ <u>&lt;size&gt;</u> ]	Get the specified coreitem.
INIT <driveno>	Initialise the catalog on the disc drive.
INT <file>	Read a sequence of command lines from the specified file. Terminate at an END command.
KILL <process>	Kill the specified process.
LIST [ <u>/CORE   /PROGRAM</u> ] <name> *	List all or selected processes, programs or coreitems.
LOAD [ <u>/&lt;coreitem&gt;</u> [ <u>/&lt;size&gt;</u> ]] { {<file>   ( <file> <params> ) } [ <u>/&lt;procname&gt;</u> ] } <sup>+</sup>	Load a coreitem from the specified file(s).
START <process>	Start the specified process.
STOP <process>	Stop the specified process.
<filename><params>	Load a utility coreitem from the specified file.

APPENDIX C, SURVEY OF ERROR MESSAGES AND EQUIVALENT ERROR NUMBERS

1       \*\*\* SYNTAX  
2       \*\*\* TOO MANY PARENTHESES  
3       \*\*\* PARAM  
4       \*\*\* END MEDIUM, FILE <filename>  
5       \*\*\* TOO MANY COMMANDS  
6       \*\*\* STATUS, FILE <filename><status>  
7       \*\*\* UNKNOWN, FILE <filename>  
8       \*\*\* RESERVATION, FILE <filename>  
9       \*\*\* COREITEM EXISTS, ITEM <itemname>  
10      \*\*\* SIZE  
11      \*\*\* COREITEM DOES NOT EXIST, ITEM <itemname>  
12      \*\*\* COREITEM NOT CLEARED, ITEM <item>  
13      \*\*\* ENTRY NOT A FILE, ENTRY <catalog entry>  
14      \*\*\* STATUS, DEVICE <device name><status>  
15      \*\*\* NOT ALLOWED  
  
16      \*\*\* NO SPACE FOR PAGES, FILE <filename>  
17      \*\*\* ILLEGAL PROGRAM, FILE <filename>  
18      \*\*\* SIZE ERROR, FILE <filename>  
19      \*\*\* CHECKSUM ERROR, FILE <filename>  
20      \*\*\* VIRTUAL ADDRESS ERROR, FILE <filename>  
21      \*\*\* PROCESS DOES NOT EXIST, PROCESS <process>  
22      \*\*\* SYSTEM ERROR <number>  
23      \*\*\* PROCESS EXISTS' PROCESS <process>



