Title:

DOMUS, User's Guide, Part 1
Version 1

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 43-GL 5643
Edition: December 1977
Author: Philippe Gaugin

Keywords:

DOMUS, MUS, Operating System, Loader, Disc

Abstract:

This manual describes the disc operating system DOMUS for the RC3600 line of computers. 56 pages.

CONTENTS

REFERENCES

[1] MUS-SYSTEM INTRODUCTION (I) and
MUS PROGRAMMER'S GUIDE (II).

Keywords: Multiprogramming, monitor, device handling, i/o-utility, record i/o, operator communication, operating system.

Abstract: (I) This manual is intended as an introduction guide to the Multi-programming Utility System.

(II) The manual is mainly intended for readers who are going to use the system. The user is assumed to be familiar with the general principles of the system as well as with the assembler language.

[2] DOMUS System Programmer's Guide.

Keywords: MUS, Operating System, Loader, disc.

Abstract: This manual describes the interface between assembly programs and DOMUS.

[3] RC 3600 PAGING SYSTEM
SYSTEM PROGRAMMER'S GUIDE

Keywords: MUS, Paging System, Virtual Memory, Address Mapping.

Abstract: This manual describes how to use the RC 3600 paging system from assembly programs under the MUS-system.

[4] RC 3600 FILE SYSTEM
SYSTEM PROGRAMMER'S GUIDE

Keywords: File system, catalog, area process, cat. 76.

Abstract: This Manual describes how to use the RC 3600 file system from Assembler programs. The user must be familiar with the MUS system.

[5] MUSIL

Keywords: RC 3600 MUS System Software, Programming Language.

Abstract: Syntax Rules for MUSIL language. Description of standard procedures. Explanation of I/O handling.

[6] MUSIL COMPILER
Operators Guide

Keywords: RC 3600 Musil Compiler.

Abstract: Compiler Guide, Operators Guide.

[7] DOMUS User's Guide PART II.

Keywords: DOMUS, MUS, Operating System, Loader, disc.

Abstract: This manual describes the utility system for the disc operating system DOMUS for the RC 3600 line of computers.

[8] MUSIL TEXT EDITOR

1. INTRODUCTION

The Disc Operating Multiprogramming Utility System, DOMUS, can be used with any RC 3600 computer of 32 Kb or larger memory, together with any combination of discs and an operator device, e.g. a teletype.

The main features of DOMUS are:

- Parallel processing including interprocess communication and interrupt processing.

- A strong framework for i/o processing, both on character level and on record oriented level.

- The operating system takes care of core storage allocation, program load from disc files, process creation and removal.

- The operating system itself has only a minor part resident in core, the major part residing on a disc. Also user programs can use this facility.

- Easy operation of the operating system by a human operator or by user programs sending internal commands to the operating system.

- Support for the MUSIL TEXT EDITOR and the MUSIL COMPILER and other utilities, and support for driver programs for all hardware modules of the RC 3600 system.

1.1 Terminology

address	An address may be a word address, which is a 15 bit unsigned integer, corresponding to a physical address in core store. Or it may be a byteaddress, which is a word address left shifted one and with a one added in bit 15 if the byte addressed within the word is to the right.
bit	A computer word consists of 16 bits, numbered from left to right: B0, B1, B2, ... B15.
byte	A computer word is regarded as two 8 bit bytes. The left one bit0 to bit7 has a even address and the right one bit8 to bit15 an odd address.
character	A character is a byte. The common alphabet within the system is the ASCII alphabet see appendix A.
text	A text is a sequence of characters. Starting at a byte address and containing in a left to right packing. A text is terminated by a Null character with byte value zero.
descriptor	A collection of information, which describes an object, is called a descriptor. Descriptors are found as part of items and as part of zones.
item	An item is a core area, which is headed by a descriptor, the first part of which usually has a standard layout. This ensures that an item always may be in some chain and possibly also in

a queue. The first words of an item contains the fields:

next:	next item in a queue
prev:	previous item in a queue
chain:	next item in a chain
size:	the size of the core area of item
name:	(3 words) A text identifying the item.

field — A field is a displacement, which identifies a piece of information within a descriptor. Some important fields are predefined in the system assembler, and/or in the musil compiler.

chain — (linked linear list). A chain consists of a chain head and a number of chain elements. The head and each element point at the next item in the chain, the last element equals zero.

queue — (doubly linked cyclical linear list). A queue consists of one or more queue elements. One of the elements is the queue head. A queue element consists of two consecutive words pointing at the next element in the queue and the previous element in the queue respectively.

When a queue is empty the head points at itself. When an element is not in a queue it normally points at itself.

length — The term length is used to express the number of bytes contained in some core area.

size	The term size is used to express the number of words contained in some core area.
program	A collection of instructions and data which may be executed or accessed by one ore more processes.
process	A sequential execution of programs under control of the monitor. All information about a process is collected in a process descriptor.
monitor	The nucleus of the system which implements multiprogramming, i.e. a parallel execution of several processes on a single processor.
device	A collection of units which can receive data from the processor or transmit data to the processor, often in parallel with the execution of computer instructions.
driver	A process executing a driver program in order to central i/o to a device.
disc	Any random access storage unit connected to the computer.
drive	A disc unit station in the system. All drives are numbered from zero to a maximum and are administrated by the cat process.
file	A logical collection of data residing on a disc having a name (discfile). Sometimes we shall denote a roll of paper tape or a collection of data between to tape marks on a magtape reel as a file too.
zone	A collection of information and associated storage areas neccessary to perform operation on files and devices.

1.2 Files

Files residing on discs are identified with a name consisting of 5 ascii characters. DOMUS only accepts filenames beginning with a letter and continued by letters and digits, only the first 5 characters being significant.

There exists no explicit type of the different files in the system, they may however be classified in 4 different types:

- Text Files

  Consist of a sequence of ascii characters, the NULL char (a zero byte) being totally ignored, terminated by an EM char (byte value 25) or the physical end of medium.

- Relocatable Binary Files

  Contain a program/process which can be loaded and started by DOMUS. The file is terminated by the physical end of medium.

- Absolute Binary Files

  Contain a stand-alone program which can be bootstrapped by DOMUS. The file is terminated by the physical end of medium.

- Data Files

  Contain data produced by user programs.
  These files are of no interest for DOMUS.

1.3 Drives

Programs and processes are loaded from the files residing on the disc mounted on the current drive.

After system bootstrap the current drive is set to a drive of the master device. This drive is called the master drive. The disc mounted on the master drive usually contains drivers for alle devices connected to the computer, the system utilities and the commonly used programs.

1.4 The Operating System Process S

The DOMUS-system consists of the following basic software components:

Monitor
Utility Procedures
Basic i/o
Character i/o
Record i/o
Paging System
Master device driver
Operator device driver
File Management System
Operating System S

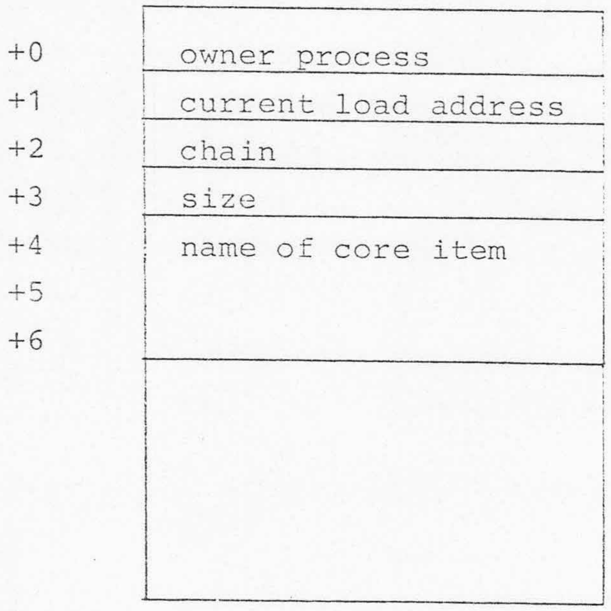
The operating system S takes care of core storage management, program/process load from disc files, program/process removal. It executes commands keyed in on the operator device or sent to the process from another process in the system.

1.5 Core Storage Management

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



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 loaded 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 the name of the core item.

These core items are classified into 3 classes.

Free core items:	The item is not owned by any process, i.e. the core is not used.
Used core item:	The item is owned by a process, which may use it for any purpose, e.g. loading of programs/processes, storing data etc.
Utility core item:	The item is automatically allocated to the owner process during a load, but the item cannot be used explicitly.

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.

$$1)\quad \left.\begin{array}{l} p\ \underline{\text{in}}\ C \quad \equiv \\ C\ \underline{\text{contains}}\ p: \end{array}\right\} \quad \text{p lies inside the core item C}$$

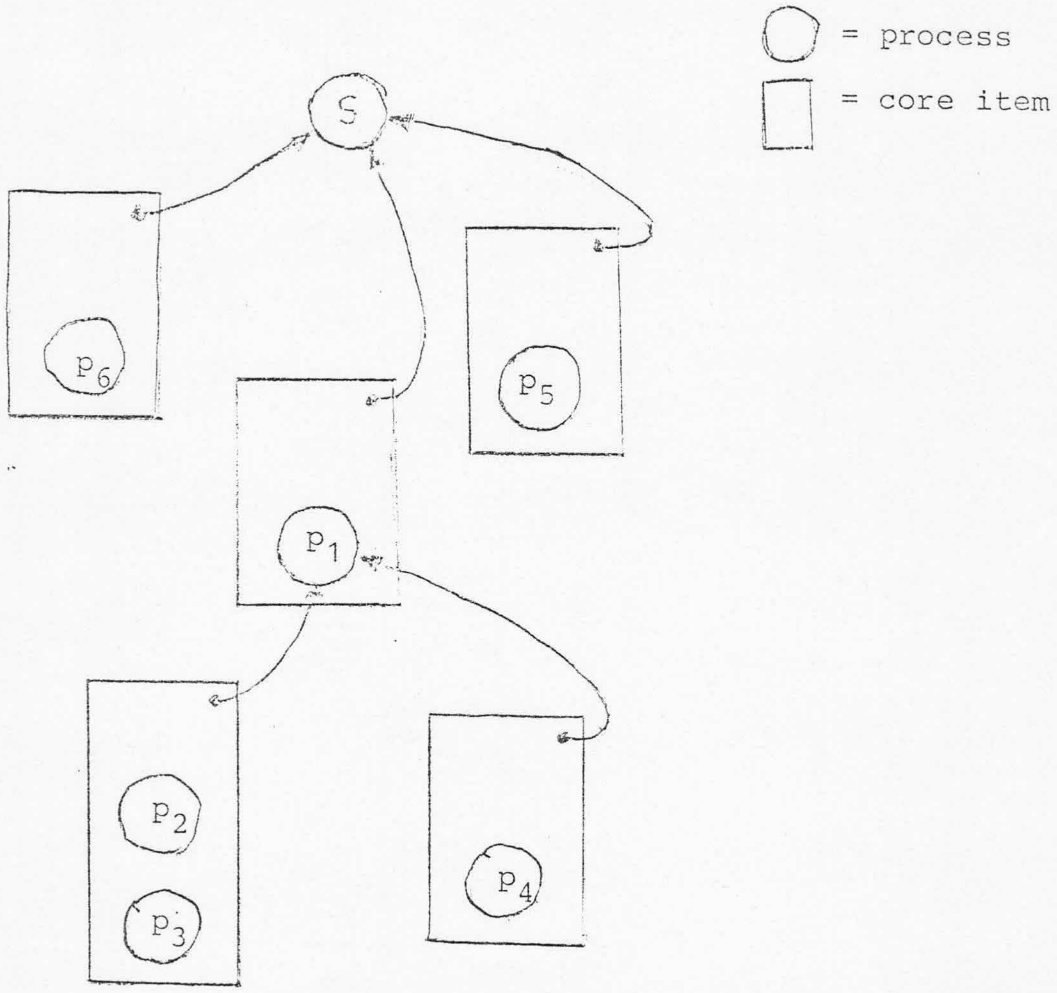
$$2)\quad \left.\begin{array}{l} p\ \underline{\text{owns}}\ C \quad \equiv \\ C\ \underline{\text{owned by}}\ p: \end{array}\right\} \quad \text{C's owner is p}$$

1.5.2 Process Hierachy

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

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

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.



2. WORKING CYCLE OF S

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

1. Console command

A human operator has keyed in a command to be executed.

2. Internal command

A process in the system has sent a message to S containing a sequence of commands to be executed.

3. Internal request

A process in the system has sent a message to S, wanting S to kill itself and execute a sequence of commands.

Now S starts to execute the first command. The command may force S to read more commands from the operator device or to read more commands from a file on the current drive. Commands may also change the current drive to another drive in the system. All commands are executed in a strict sequentiel manner, until there are no more commands to execute. Some commands may be treated differently depending on whether the execution was initiated via the console or not. When the last command has been executed, S returns to the idle state, resets the current drive to the master drive and waits for the next event.

As the S-functions only take "a short time" to execute, you may always espect S to be ready to accept a console command, i.e. ready to perform a quick operator invention.

If an error occurs while executing a console command S will print an errormessage on the operator device. If an error occurs while S is executing an internal command or request, S will send an appropriate answer containing information about the error. When S receives an internal request containing no commands it will just kill the process and print the message: FINIS <process name> on the operator device.

2.1 The operating process

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 accepted if the sender of the message is a child of S, and the operating process is then defined as S itself.

Cenerally 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 function, however, violates this rule.

3. THE S COMMAND LANGUAGE

The following metalinguistic symbols are used, in the description of the S command lauguage:

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 occurence 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 occurrencies of the preceeding variable, whereas the superscription $^+$ means one or more occurrencies. The brackets [and] indicates an optional string.

3.1 Basic elements

Syntax:

```
<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 charactes LF, VT, FF or CR
```

3.2 Numbers

Syntax:

```
<integer>         ::= <digit>+
<sign>            ::= +|-
<radix>           ::= <integer>'
<numbers>         ::= [<sign>][<radix>]<integer>
```

Semantics:

A number represents a 16 bit integer quantum. If no sign is present the number is regarded as positive.
If - is present the two's complement of the number is used.
If no radix is present the integer is interpreted as a decimal number. The radix denotes that the following integer should be converted digit by digit as follows:
number:=number*radix+digit. All numbers are treated modulo 2^{16}.
A number is terminated by the first non digit following the number.

3.3 Texts

Syntax:

<text> ::= '<any character except'>'

Semantics:

A text represents a sequence of charactes of any length.

3.4 Names

Syntax:

<name> ::= <letter> $\{$<letter>|<digit>$\}^*$

Semantics:

A name is used to identify a file, a process, a program, an S-function or is has a special meaning depending on the S-function or utility program using the name.
Only the first 5 charactes in the name are significant. The name is terminated by the first non letter or digit following the name.

3.5 Items

Syntax:

```
<item>             ::= <name>|<number>|<text>|
                       <dummy item>|<composite item>
<composite item> ::= (<item> {[<sep>]<item>}* )
<dummy item>       ::= *
<sep>              ::= . | / | : | = |,
```

Semantics:

Items are the fundamental entities forming S-commands. Each item is treated as a unit by the command intepreter of S, and any separators appearing in front of an item are related to that item. Items and the preceeding separator (if any) are packed into a internal form by S. Consult ref [2] for further information on the internal representation of items.
The dummy item denotes the absense of a paramter. The composite item denotes a record of parameters consideres as one parameter.

3.6 S-Commands

Syntax:

```
<command>          ::= <S-function> {[<sep>]<item>}* <nl>|
                       <filename> {[<sep>]<item> }* <nl>|
                       <nl>

<S-function>       ::= <name>
<filename>         ::= <name>
```

Semantics:

If the first item is the name of an S-function, that function is executed with the remaining items considered as parameters to the S-function. Else the filename is looked up in the directory on the current drive. If the file exists and contains a program, the program is loaded with the items as paramters, and S itself continues.

3.7 Comments, Blanks, Blind characters.

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

4. S-FUNCTIONS

S-functions are executed by S itself. If it is impossible to execute the function properly, an errormessage is printed on the operator device if the S-function was executed as a console command. If the execution was initiated via an internal command or request an answer containing the error cause is returned. See ref [2] for further information. When an error occurs, the function is aborted and S skips remaining commands in a command sequence and goes into the idle state.

Errormessages consist of 3 components:

1. An errorcause
2. Possibly a name
3. Possibly a number

When printing error messages on the operator device the messages has the format:

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

where the text explains the error cause. A list of errormessages are presented in appendix C.

In the following each S-function is listed using this scheme:

S-FUNCTION:	name of S-function.
FORMAT:	format of commands activating the function.
FUNCTION:	explanation of the function when executed as a console command.
EXAMPLES:	one or more examples of the use.

ERRORS:	list of errormessages that may appear and an equivalent errornumber. The following error-messages may appear in all contexts:

***SYNTAX	(1)
***TOO MANY PARENTHESIS	(2)
***END MEDIUM, FILE <filename or operator device>	(4)
***TOO MANY COMMANDS	(5)
***SYSTEM ERROR <number>	(22)

INTERNAL EXECUTION:	Modifications, if any to the above explanation of the function, when the command is executed as an internal command or request.

S-FUNCTION: BEGIN

FORMAT: BEGIN

FUNCTION: The function causes S to read a sequence of commands from the operator device. The reading continues until an END command has been read. Then S starts to execute the commands one by one up to and including the first END, INT or another BEGIN command. Note that no internal commands or requests can interrupt the above sequence, before an END command has been executed.

EXAMPLES:

```
BEGIN  !LOAD AND LIST PTR DRIVER!
   LOAD PTR
   LIST PTR
END
```

ERRORS: ***PARAM (3)

INTERNAL EXECUTION: The command is dummy.

S-FUNCTION:	BOOT
FORMAT:	BOOT <filename>
FUNCTION:	The function loads an absolute binary program from the file specified. The file should reside on the current drive. The command terminates the normal execution of the DOMUS system and replaces it with a stand alone program, perhaps being another MUS system or the like.
EXAMPLES:	BOOT BACRES !terminate the DOMUS system and bootstrap the program BACRES!

ERRORS:

```
***PARAM                                  (3)
***STATUS, FILE <filename> <status>       (6)
***UNKNOWN, FILE <filename>               (7)
***RESERVATION, FILE <filename>           (8)
***ILLEGAL PROGRAM, FILE <filename>       (17)
   The file does not contain a absolute
   binary program.
***SIZE ERROR, FILE <filename>            (18)
   No more core available during the
   creation of the absolute core image
   of the program.
***CHECKSUM ERROR, FILE <filename>        (19)
```

INTERNAL EXECUTION: As described above.

S-FUNCTION:	BREAK
FORMAT:	BREAK <process name>
FUNCTION:	The function performs a break on the process given by <process name>. The processes in the basic system including S itself are not allowed to be breaked. The process is started in its break address with errornumber = 2.
EXAMPLES:	BREAK MAIN
ERRORS:	***PARAM (3) ***NOT ALLOWED (15) ***PROCESS DOES NOT EXIST, PROCESS <process name> (21)
INTERNAL EXECUTION:	As described above.

S-FUNCTION: CLEAN

FORMAT: CLEAN <process name>

FUNCTION: The function performs a stopprocess and a clean-process on the process given by <process name>. The processes in the basic system including S itself are not allowed be cleaned. The function should not be used unless you know the consequences of cleaning processes, and should only be used during program debugging.

EXAMPLES: CLEAN MAIN

ERRORS:

```
***PARAM                                      (3)
***NOT ALLOWED                                (15)
***PROCESS DOES NOT EXIST, PROCESS
   <process name>                             (21)
```

INTERNAL EXECUTION: As described above.

S-FUNCTION:	CLEAR
FORMAT:	CLEAR [<core item name>]
FUNCTION:	The function clears either the specified core item or if no core item name is present, all utility core items owned by the operating process. Clear specified core item: CLEAR <core item name>clears the core item specified in the following way: First all processes inside the core item are killed (see KILL), causing the load address to be adjusted to the lowest possible address. If the core item is a utility core item, the core item is returned to the pool of free items. It is not allowed to clear a core item not owned by the operating process. Clear all utility items: CLEAR acts as a sequence of clears on alle utility core items owned by the operating process.
EXAMPLES:	CLEAR A
ERRORS:	***PARAM (3) ***COREITEM DOES NOT EXIST, ITEM <core item name> (11) ***NOT ALLOWED (15)
INTERNAL EXECUTION:	As described above.

S-FUNCTION:	DRIVE
FORMAT:	DRIVE <driveno>
FUNCTION:	The function selects the drive specified by driveno as the current drive. This drive remains the current drive for all succeding executions of commands within a command sequence, i.e. until the END command or another DRIVE command is executed. All files used in INT, LOAD and BOOT commands are supposed to reside on the current drive. The connection between drives and the physical environment on an installation is fixed at system generation time. The DRIVE command does not check if the selected drive is operable.

EXAMPLES: Let file SCOM on drive 2 have the following contents:

```
    LOAD  A B C
END
```

Then the files A, B and C on that drive could be loaded by the commands:

```
BEGIN
    DRIVE 2
    INT SCOM
END
```

ERRORS:	***PARAM	(3)
INTERNAL EXECUTION:	As described above.	

S-FUNCTION:	END
FORMAT:	END
FUNCTION:	The command acts as terminator for a sequence of commands, when S is reading, either from operator device caused by the BEGIN command, or from a disc file caused by the INT command. When the command is executed S returns to its idle state.
EXAMPLES:	File SCOM contains the following S commands LOAD X Y LIST X END These commands would be executed if you enter the console command: INT SCOM
ERRORS:	none.
INTERNAL EXECUTION:	As described above.

S-FUNCTION:	FREE
FORMAT:	FREE <core item name>
FUNCTION:	The function returnes the core item specified to the pool of free core. The core item should be owned by the operating process, and it may not contain any process.
EXAMPLE:	FREE A
ERRORS:	***PARAM (3)
	***COREITEM DOES NOT EXIST, ITEM <core item name> (11)
	***COREITEM NOT CLEARED, ITEM <core item name> (12) The item contains a process descriptor. Clear the item, using the CLEAR command.
	***NOT ALLOWED (15) The owner of the item is not the operating process.
INTERNAL EXECUTION:	As described above.

S-FUNCTION: GET

FORMAT: GET <core item name> [<size>]

FUNCTION: The function allocates a core item with the specified name. If size is specified the function allocates a core item with size equal to the number of words specified, or larger using a first fit strategy, else the maximum core item is allocated. The owner of the core item is set to the operating process and the current load address is set to the first word after the core item head. After a succesful allocation, the core item is a used core item owned by the operating process.

EXAMPLES:

```
GET A 512
LIST/CORE A
A  17334  1000  17343  S
```

ERRORS:

```
***PARAM                                      (3)
***COREITEM EXISTS, ITEM <core item
   name>                                      (9)
   It is impossible to get an item with
   the name of an already existing core
   item name.
***SIZE                                       (10)
   It is impossible to get an item of
   the specified size.
```

INTERNAL EXECUTION: As described above.

S-FUNCTION:	INIT
FORMAT:	INIT <drive no>
FUNCTION:	The function sends an init catalog message to the file handler containing the specified driveno, see ref [4]. The master drive is always operable, but the other drives in the system cannot be used before the INIT command has been executed for that drive.
EXAMPLES:	`INIT  1    ! Initialise drive 1!`
ERRORS:	***PARAM
	***STATUS, DEVICE CAT <status>
	Drive not operable
INTERNAL EXECUTION:	As described above.

S-FUNCTION: INT

FORMAT: INT <filename>

FUNCTION: The function causes S to read a sequence of commands from the file specified. The file should reside on the current drive. The reading continues until an END command has been read. Then S starts to execute the commands one by one up to and including the first END, INT or BEGIN command. Note that no other commands can interrupt the above sequence before an END command has been executed. By putting INT commands in a command file you can chain a number of command files. If you do this make sure that the chain is finite.

EXAMPLES: File SCOM1 contains the commands

```
    LOAD  A  B
    LIST  A  B
    INT   SCOM2
END
```

File SCOM2 contains the commands

```
    LOAD  C
    LIST/CORE C
END
```

When you enter the command INT SCOM1 the files A and B will be loaded and the processes A and B will be listed. Then the commands in SCOM2 will be executed, as shown below:

```
INT SCOM1
A  17145  A
B  22022  B
C  22113  1011 ...... S
```

ERRORS:	***PARAM	(3)
	***STATUS, FILE <filename>	(6)
	***UNKNOWN, FILE <filename>	(7)
	***RESERVATION, FILE <filename>	(8)
	***ENTRY NOT A FILE, ENTRY <filename>	(13)
INTERNAL EXECUTION:	As described above.	

S-FUNCTION:	KILL
FORMAT:	KILL <process name>
FUNCTION:	The function kills the specified process in the following way. All core items owned by the process are cleared (see CLEAR function) and are returned to the pool of free items. Then the process itself is removed, and if the surrounding core item is a utility item, the item is also returned to the pool of free items, otherwise the load address of the surrounding core item is adjusted. The adjustment of the load address is done if there are no processes in the core item with a process description address higher than the process description address of the process to be removed, and the load address is reset to the value used when the process was originally loaded. It is only allowed to remove a process being a child of the operating process.
EXAMPLES:	KILL MAIN
ERRORS:	***PARAM (3)
	***NOT ALLOWED (15)
	***PROCESS DOES NOT EXIST, PROCESS <process> (21)
INTERNAL EXECUTION:	As described above.

S-FUNCTION: LIST

FORMAT: LIST [/PROGRAM|/CORE] <name>*

FUNCTION: The function lists items in one of three chains, the process chain, the program chain or the core item chain. The items are listed on the operator device. The function lists selected items or, if no <name> is present, all items in the chain. All numbers are printed in octal.

Process list:
LIST <name>* selects the process chain, and the output has the following format:

<name> <address> [@] [<core item name>]

<name> is the name of the process.
<address> is the process description address of the process.

@ indicates that the process is a driver process and that no process has reserved the driver.
<core item name>is the name of the surrounding core item, blank if the process reside in the basic system.

Program list:
LIST/PROGRAM <name>* selects the program chain, and the output has the following format:

<name> <address>

<name> is the name of the program.
<address>is the address of the program head, equal to the relocatable base of the program when it was loaded.

Core item list:
LIST/CORE <name>* selects the core item chain, and the output has the following format:

```
[<name>]<address><size>
        <current load address><owner process name>
```

<name> is the name of the core item, blank if the core item is free.
<address>is the address of the core item descriptor.
<size> is the size of the core item.
<current load address> is the relocatable base for the next load into this core item, or it is printed as ...... if the item is a utility item.
<owner process name> is the name of the process owning the core item.

Generally:
If a name appears in the name list: <name>* and it does not exist in the selected chain, no output is generated for that name.

EXAMPLES:

```
LIST  LPT  SPT
LPT   40377     SPT
!NOTE THE PROCESS SPT DID NOT EXIST!
LIST/CORE SPT
SPT   40370  276 ...... S
```

ERRORS: ***PARAM (3)

INTERNAL EXECUTION: The command is dummy.

S-FUNCTION: LOAD

FORMAT:

```
LOAD [/<core item name>[/<size>]]
{{ <filename>| (<filename> <params>)}
        [/<process name>]}+
```

FUNCTION: The basic function of S, making it possible to load programs and processes from files. The function loads a list of files specified by <file name> or (<file name> <params>).
If the program accept parameters, the first form results in <filename> being the only parameter, and the second form results in <filename> <params> being the parameters. If the file contains a process descriptor the address of the parameters are delivered to the process through an accumulator, see [2] for further information.
Each process loaded can be renamed by adding the /<process name> to the filename. Otherwise the process name in the process descriptor is used.

Load into free core:

```
LOAD {{ <filename>|(<filename><params>)}
        [/<process name>]}+
```

When using this format the load will take place in the largest free core item in the core item chain. When each file has been loaded and the parameters are appended to the program, the core item is cut to the minimal size still containing the program, the name of the core item is set to the name of the file from where the program was loaded, the load address is set to zero and the owner to the operating process.
Thus a utility item is created.

Load into a specific core item:

```
LOAD / <core item name>[/<size>]
     {{ <file name>!(<file name><params>) }
      [/<process name>] }+
```

When using this format, each file is loaded into the specified core item, starting at the current load address of the core item. When each file has been loaded and the parameters are appended to the program, the current load address is adjusted.

The load can never exceed the core storage occupied by the core item. The size parameter forces S to check that the load does not overwrite the core storage behind the first <size> locations of the core item. So you can protect information in the core item from being destroyed by a load. It is only allowed to load into a core item owned by the operating process.

EXAMPLES:	LOAD PTR	!load of a paper tape reader driver!
	LOAD SPT/LPT	!load of a serial printer driver and renaming the process to LPT!
	LOAD PTP (PIP 1 2 3)	!load of a paper tape punch driver and a utility program with parameters!
	LOAD/A PPP	!load the program PPP into the core item A!

ERRORS:

***PARAM (3)
Errors in the format. Note that the format is checked before any load.

***STATUS, FILE <file name> <status> (6)

***UNKNOWN, FILE <file name> (7)

***RESERVATION, FILE <file name> (8)

***CORE ITEM DOES NOT EXIST, ITEM <core item name> (11)

***ENTRY NOT A FILE, ENTRY <catalog entry> (13)

***NOT ALLOWED (15)

***NO SPACE FOR PAGES, FILE <file name> (16)
The disc file used for saving the pages of paged programs is filled during the load of the program on file <file name>. Usually a system generation error.

***ILLEGAL PROGRAM, FILE <file name> (17)
The file <file name> does not contain a relocatable binary program.

***SIZE ERROR, FILE <file name> (18)
No more core available for the load of the program on file <file name>.

***CHECKSUM ERROR, FILE <file name> (19)
Checksum error during load of the program on file <file name>

***VIRTUAL ADDRESS ERROR, FILE <file name> (20)
The program contains an illegal virtual address or an errorneous page map.

***PROCESS EXISTS, PROCESS <filename> (23)
The file cannot be loaded because a process exists with the same name as the filename

INTERNAL EXECUTION: As described above.

S-FUNCTION:	START
FORMAT:	START <process name>
FUNCTION:	The function performs a startprocess on the process given by <process name>. The processes in the basic system including S itself are not allowed to be started.
EXAMPLES:	START MAIN
ERRORS:	***PARAM (3)
	***NOT ALLOWED (15)
	***PROCESS DOES NOT EXIST, PROCESS <process name> (21)
INTERNAL EXECUTION:	As described above.

S-FUNCTION:	STOP
FORMAT:	STOP <process name>
FUNCTION:	The function performs a stopprocess on the process given by <process name>. The processes in the basic system including S itself are not allowed to be stopped.
EXAMPLES:	STOP MAIN
ERRORS:	***PARAM (3)
	***NOT ALLOWED (15)
	***PROCESS DOES NOT EXIST, PROCESS <process name> (21)
INTERNAL EXECUTION:	As described above.

S-FUNCTION:	Utility program load
FORMAT:	<file name>[<params>]
FUNCTION:	if <file name> is not identical to any other S-function, the command works as the command: LOAD (<file name> [<params>]). See LOAD for futher information
EXAMPLES:	PRINT PIP

ERRORS:

```
*** PARAM                                   (3)
*** STATUS, FILE <file name><status>        (6)
*** UNKNOWN, FILE <file name>               (7)
*** RESERVATION, FILE <file name>           (8)
*** ENTRY NOT A FILE, ENTRY <file name>    (13)
*** NO SPACE FOR PAGES                     (16)
    The disc file used for saving the
    pages af paged program is filled du-
    ring load of the program on file
    <file name>. Usually a system genera-
    tion error.
*** ILLEGAL PROGRAM, FILE <file name>      (17)
    The file <file name> does not con-
    tain a relocatable binary program.
*** SIZE ERROR, FILE <file name>           (18)
    No more core available for the load
    of the program on file <fiel name>
*** CHECKSUM ERROR, FILE <file name>       (19)
    Checksum error during load of the
    program on file <file name>
***VIRTUAL ADDRESS ERROR, FILE <file -
    name>                                  (21)
    The program contains an illegal vir-
    tual address or an errorneous page
    map.
```

***PROCESS EXISTS, PROCESS <filename> (23)
The file cannot be loaded because a process exists with the same name as the filename.

INTERNAL EXECUTION:	As described above.

APPENDIX A, CHARACTER SET USED BY DOMUS

V	N	C	V	N	C	V	N	C	V	N	C
0	NUL	blind	32	SP	blank	64	@	ill.	96	\	ill.
1	SOH	ill.	33	!		65	A		97	a	
2	STX	ill.	34	"	ill.	66	B		98	b	
3	ETX	ill.	35	#	ill.	67	C		99	c	
4	EOT	ill.	36	$		68	D		100	d	
5	ENQ	ill.	37	%	ill.	69	E		101	e	
6	ACK	ill.	38	&	ill.	70	F		102	f	
7	BEL	ill.	39	'		71	G		103	g	
8	BS	ill.	40	(		72	H		104	h	
9	HT	blank	41	)		73	I		105	i	
10	LF	nl.	42	*		74	J		106	j	
11	VT	nl.	43	+		75	K		107	k	
12	FF	nl.	44	,		76	L		108	l	
13	CR	nl.	45	-		77	M		109	m	
14	SO	ill.	46	.		78	N		110	n	
15	SI	ill.	47	/		79	O		111	o	
16	DLE	ill.	48	0		80	P		112	p	
17	DC1	ill.	49	1		81	Q		113	q	
18	DC2	ill.	50	2		82	R		114	r	
19	DC3	ill.	51	3		83	S		115	s	
20	DC4	ill.	52	4		84	T		116	t	
21	NAK	ill.	53	5		85	U		117	u	
22	SYN	ill.	54	6		86	V		118	v	
23	ETB	ill.	55	7		87	W		119	w	
24	CAN	ill.	56	8		88	X		120	x	
25	EM	em.	57	9		89	Y		121	y	
26	SUB	ill.	58	:		90	Z		122	z	
27	ESC	ill.	59	;	ill.	91	Æ		123	æ	
28	FS	ill.	60	<	ill.	92	Ø		124	ø	
29	GS	ill.	61	=		93	Å		125	å	
30	RS	ill.	62	>	ill.	94	↑	ill.	126	~	ill.
31	US	ill.	63	?	ill.	95	←	ill.	127	DEL	ill.

V = 7 bit value of character
N = name of character
C = comments

If no comments the characters are recognized by DOMUS according to the syntax of S commands.

ill. means that the character is illegal outside texts and comments.
em. means that the character signifies end of medium.
nl. means that the character is regarded as the terminator of a command line.

APPENDIX B, SURVEY OF S-COMMANDS

BEGIN	Read a sequence of command lines from the operator device. Terminate at an END command.
BOOT <file name>	Load an absolute binary program.
BREAK <process name>	Break the specified process.
CLEAN <process name>	Stop and clean the specified process.
CLEAR <core item name>	Clear specified core item, or all utility items.
DRIVE <driveno>	Select the specified drive as the current drive.
END	Terminate a sequence of commands and execute these commands.
FREE <core item name>	Free the specified core item.
GET <core item name> [<size>]	Get the specified core item.
INIT <driveno>	Initialise the catalog on the specified drive.
INT <file name>	Read a sequence of command lines from the specified file. Terminate at an END command.
KILL <process name>	Kill the specified process.

LIST [/PROGRAM|/CORE] <name>*	List all or selected processes, programs or core items.
LOAD [/<core item name>[/<size>]] {{<file name>|(<file name><params>)}[/<process name>]}+	Load the specified file(s).
START <process name>	Start the specified process.
STOP <process name>	Stop the specified process.
<filename>[<params>]	Load the specified file.

APPENDIX C, SURVEY OF ERROR MESSAGES AND NUMBERS

1 ***SYNTAX

The command to S does not fulfill the syntax described in appendix A (all functions).

2 ***TOO MANY PARENTHESES

The command to S contains too many parentheses. Implementation restriction (all functions).

3 ***PARAM

The selected S-functions cannot interprete the parameters in a meaningful way (BEGIN, BOOT, BREAK, CLEAN, CLEAR, FREE, GET, INIT, INT, KILL, LIST, LOAD, START, STOP, Utility program load).

4 ***END MEDIUM FILE <filename>

The reading from a file is terminated due to physical end of medium on a file, or an end medium character during command reading (all functions).

5 ***TOO MANY COMMANDS

The command (sequence) too long, because there is not enough free core storage available, or because of an implementation restriction (all functions).

6 ***STATUS, FILE <filename> <status>

The reading from a file is terminated due to a status error from the file management system. The octal status is shown (BOOT, INT, LOAD, Utility program load).

7 ***UNKNOWN, FILE <filename>

The filename does not exist in the directory on the current drive (BOOT, INT, LOAD, Utility program load).

8 ***RESERVATION, FILE <filename>
The file is reserved for exclusive use by another process in the system (BOOT, INT, LOAD, Utility program load).

9 ***COREITEM EXISTS, ITEM <core item name>
The core item should not exist in order to execute an S-function (GET).

10 ***SIZE
Not enough core storage available to execute an S-function (GET).

11 ***COREITEM DOES NOT EXIST, ITEM <core item name>
The core item should exist in order to execute an S-function (CLEAR, FREE, LOAD).

12 ***COREITEM NOT CLEARED, ITEM <core item name>
The core item contains some processes, when it should not (FREE).

13 ***ENTRY NOT A FILE, ENTRY <filename>
The filename does exist in the directory on the current drive, but it is not a disc file (BOOT, INT, LOAD, Utility program load).

14 ***STATUS, DEVICE <device name> <status>
Communication trouble with a process. The octal status is shown (INIT).

15 ***NOT ALLOWED
The execution violates some restrictions. Check with description of S-functions (BREAK, CLEAN, CLEAR, FREE, KILL, LOAD, START, STOP).

16 ***NO SPACE FOR PAGES, FILE <filename>
(LOAD, Utility program load)

17 ***ILLEGAL PROGRAM, FILE <filename>
A relocatable or absolute binary program does not fulfil the conventions for these type of files (BOOT, LOAD, Utility program load).

18 ***SIZE ERROR, FILE <filename>
A program is to big to be loaded at the moment. Try to FREE more core (BOOT, LOAD, Utility program load).

19 ***CHECKSUM ERROR, FILE <filename>
Checksum error in a relocatable or absolute binary file (BOOT, LOAD, Utility program load).

20 ***VIRTUAL ADDRESS ERROR, FILE <filename>
Illegal coding in paged programs (LOAD, Utility program load).

21 ***PROCESS DOES NOT EXIST, PROCESS <process>
The process should exist in order to execute an S-function (BREAK, CLEAN, KILL, START, STOP).

22 *** SYSTEM ERROR <number>
The S process cannot execute the command. See the list of system errors in appendix D (all functions).

23 ***PROCESS EXISTS, PROCESS <process>
The process should not exist in order to execute an S-function (LOAD, utility program load)

***BREAK <cause> <ac1>
The S process has been breaked. Malfunction of the system. The message appears on the teletype. The system may fail to operate properly.

APPENDIX D, SYSTEM ERROR MESSAGES

The format of a system error message is:
***SYSTEM ERROR <number> where the number refers to the following list:

Bootstrap errors

The error occurs during the bootstrap of the DOMUS system

1	Operator device malfunction
2	Master drive undefined
3	Master device malfunction
4	File management system malfunction
5	Paging file error
6	System configuration error
7	Error message file error
10-16	System malfunction

Runtime errors

21	Internal request error. An error occurs after the sending process is removed. Non fatal error.
22	DOMUS stack overflow. An implementation restriction has been violated. Fatal error, the system may fail to operate properly.
24	Error message file error. Fatal error, the system may fail to operate properly.
25	Core storage structure destroyed. Fatal error, the system may fail to operate properly.

APPENDIX E, SYSTEM BOOTSTRAP

The DOMUS system is usually bootstrapped using the master drive of the system. The disc should be initialized using the program SYSGEN, and should be mounted on the master drive. If you use the RC3652 2.4 Mb disc drive as master drive, mount the discpack in unit 0, set the frontpanel switches on the CPU to bit0 = 1, device selection switches = 73 (octal), and press the autoload-bottom. In case of other disctypes set the device selection switches as specified by RC.

The bootstrap is then placed in core and outputs the query "SYSTEM:". Input must be a bootstrap commandfile containing a list of discfiles, which are linked together generating the DOMUS-system. The normal commandfile supplied is S3600, and can be selected by typing CR og NL immediately. During input "rubout" can be used to erase last character(s) input, and any control character (CR, NL, ESC) will terminate input.

If bootstrap is successful the message:

```
          DOMUS  REV  nn.nn
```

will be output on the operator device.

If there exists a file on the master drive named SSYSI, the commands in this file will be executed as if you had entered the command INT SSYSI on the operator device. Otherwise the system will invite you to enter commands by printing a

```
          >S
```

on the operator device.

In case bootstrap is unsuccessful an errormessage is output on the operator device, and the query "SYSTEM:" is output again.

Possible errors are:

* * * NOT FOUND, FILE:<filename>
File given by <filename> is not present on the master drive.

* * * ILLEGAL BLOCK TYPE, FILE:<filename>
A relocatable block other than start, title or data is found on file <filename>.

* * * ILLEGAL FILETYPE, FILE:<filename>
File <filename> is not a bootstrap commandfile as the first name found <>"BASIS", or the file is not a datafile (Attribute entry only).

* * * CHECKSUM ERROR, FILE:<filename>
One of the relocatable blocks on file <filename> has an checksum error.

* * * SIZE ERROR <octalnumber>
Not enough space in core to load the system, <octal number> is the maximal number of words that can be used by the bootstrap program.

* * * END MEDIUM, FILE:<filename>
Physical end of medium on file <filename> has been found before logical end of file. I.e. "END" name in commandfile or relocatable start block missing in file.

* * * HARD ERROR ON DISC <octal number>
The disc is malfunctioning. <octal number> is the hardware status without modifications. Please consult hardware manual for further information.

Generation of bootstrap commandfile.

The bootstrap command file is a number of ASCII filenames describing relocatable files on the master disc. This way of describing the DOMUS-system makes it flexible and eases change of modules in the system in case of new releases or new hardware configuration.

The commandfiles are made as normal text files by the RC3600 Text Editor, but they must however obey following rules:

1) First filename has to be "BASIS", in this way telling the bootstrapprogram that the file is a bootstrap command file. The name is checked but skipped.

2) The following names should be the names of relocatable modules or drivers, which are going to be present in the DOMUS-system. Each name is describing a catalog file in the catalog on the master disc, and

3) The last name in the file must be "END" terminating the file list.

4) Filenames are separated by one or more spaces, CR or NL.

5) In order to generate a well functioning DOMUS-system following modules must be present:

   MUS - Monitor
   MUS - Utility Procedure Module
   MUS - Basic I/O Module
   MUS - Record I/O Module
   MUS - Character I/O Module
   MUSIL Interpreter

Operator Console Driver (TTY)
Master Disc Driver
MUS - Catalog System
Catalog System Description Process (CATW)
DOMUS Operating System Process
MUS - Initialization Module

It is recommended that the MUS-Monitor is taken as the first module followed by all non-process modules, and then all driver processes. The last two modules must be the DOMUS Operating System and the MUS-Initialization Module in the given order, as the Initialization Module is removed after system start, and the Operating System expects rest of free memory above own code.

The presence of the above mentioned modules is checked by the Operating System after start, and if one is missing it results in a system-error output on the operator device, and the system will halt.

APPENDIX F, SYSTEM GENERATION

A DOMUS system is installed on a disc using the program RC36-00422. The program generates a DOMUS disc using magtape og paper tapes. When loaded the program prints:

```
INPUT DEVICE (MT/PT):
```

When this question is answered with MT (magtape) or PT (paper tape) the next question is:

```
INITIALIZE CATALOG (NEW/OLD):
```

If the answer is NEW an empty catalog will be written on the disc. Then the necessary files will be created in order to use the disc as a DOMUS master disc. If the answer is OLD the disc is already a DOMUS disc, i.e. it has been used before as a DOMUS disc and the program uses the old catalog on the disc.

Now the standard device descriptors, that not already exists, will be created. Each descriptor created is verified on the operator device.

If the input device is magtape a number of files will be copied to the disc and the names are verified on the operator device.

If the input device is paper tape each tape loaded is copied to a disc file with the name specified by the operator when the question

```
FILE NAME
```

is printed. When the bootstrap program is loaded the file name specified must be BOOT.

Already existing but not writeprotected files are overwritten. If a file is writeprotected a message is printed on the operator device and the program continues with the next file.

When the program has finished the message:

```
        END SYSGEN
```

is printed and the program stops.

Now the disc is formatted in the following way:

sector 0 - 31:		used by the bootstrap system and the file management system.
sector 32 - N:		used as file space, N depends on the disc types.
file:	SYS	the catalog file, see ref. [4]
	MAP	the sector allocation map file, see ref. [4]
	SSYSP	used by S for saving pages of paged programs.
	QSYSP	used in processor expansion systems.
	SSYSE	the error message file.

Below is shown a run of a sysgen program.

```
>S

INT SYSG

>SYSG

DOMUS SYSGEN PROGRAM

INPUT DEVICE (MT/PT): MT
INITIALIZE CATALOG (NEW/OLD): NEW

THIS IMPLIES THAT ALL EXISTING FILES ARE DELETED.
CONFIRM (YES/NO): YES

ØLPT
ØSP
ØCPT
ØPTPN
ØPTP
ØPTRN
ØPTR
ØTTY
ØCTØ
ØMTØ
ØCDRN
ØCDR

MUM
MUU
MUB
MUC
MUR
INT
TT005
CAP8
```

```
MTØXX
.
.
.
.
END SYSGEN

BREAK 3
```